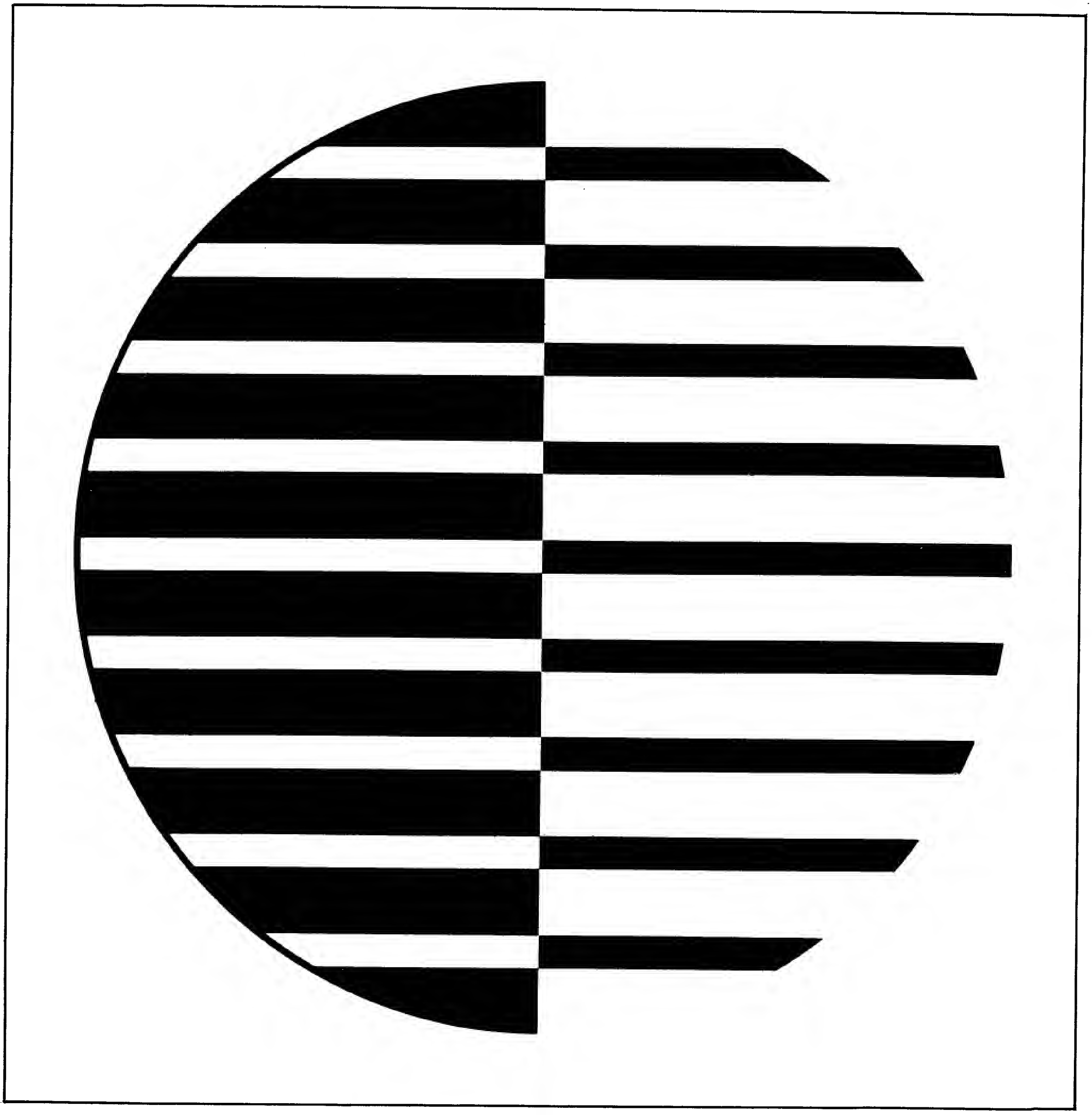
# CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

## CHIPPEWA OPERATING SYSTEM DOCUMENTATION

Volume I Preliminary Edition



| RECORD OF REVISIONS | |
|---|---|
| REVISIONS | NOTES |
| 1 | Preliminary CHIPPEWA Operating System Documentation update |
| 2 | Update May 15, 1966. |

TABLE OF CONTENTS

CONTROL DATA CORPORATION

Product Marketing Management

# NCAR ASSEMBLER

Chippewa Operating System

# I. SUMMARY

The ASCENT assembler will produce Chippewa binary decks from Ascent or Asper coding. The language specifications are as described for SSD Ascent and Asper but with some extensions and a few restrictions.

## Restrictions

1. Fortran statements may not be mixed with Ascent coding.
2. System macros are not provided.

## Extensions

1. The Ascent assembler can produce and modify COSY decks. A cosy, or COmpressed SYmbolic, deck contains all information (including comments) from the source deck but is from 1/10 to 1/5 the size of the source deck, depending on the number of comments.
2. Two consecutive field separators will terminate the variable field.
3. The variable field may contain the arithmetic operators * for multiplication and / for division.
4. Programmer macros may be used in both Ascent and Asper. Macros may call macros.
5. A VFD pseudo-op is available in Ascent.

## II. ASSEMBLER CALLS

ASCENT (L, X, PA, PC, PB, COSY).

If the first parameter is non-zero, a listing will be written on the output file.

The second parameter is inoperative. Eventually will cause a load-and-go file to be written if non-zero.

If the third parameter is non-zero, a Chippewa binary deck will be written on P80C.

If the fourth parameter is non-zero, a Cosy deck will be written on P80C.

If the fifth parameter is non-zero, a relocatable binary deck will be written on P80C.

The sixth parameter is the file from which Cosy input may be read. This may be "INPUT" or "COSY".

The nominal case is ASCENT (L, 0, 0, 0, 0, INPUT).

Examples:

1. To list only:

   ASCENT.

2. To list and punch a Chippewa binary deck

   ASSIGN CP, P80C.

   ASCENT (L, 0, PA).

3. To list and punch a Cosy deck

   ASSIGN CP, P80C.

   ASCENT (L, 0, 0, PC).

4. To list and write a cosy deck on tape 51

   ASSIGN 51, P80C.

   ASCENT (L, 0, 0, PC).

5. To insert modifications into a cosy deck on tape 51, list, and punch a Chippewa deck:

```
JOB
ASSIGN CP, P80C.
ASSIGN 51, COSY.
ASCENT (L, O, PA, O, O, COSY).
7-8-9
Mod pack ending with COSY card
6-7-8-9
```

## III. ASCENT PSEUDO-OPS

| | | | |
|---|---|---|---|
| | ASCENT | | Sets assembly mode to Ascent |
| | LIST | V1 | Suppress listing if V1 $\neq$ o |
| | SPACE | V1 | Space V1 lines |
| | EJECT | | Eject the page |
| | MACRO | NAME, V1, V2 ... | Indicates start of macro definition |
| | ENDM | | Indicates end of macro definition |
| | IFF | V1,V2,V3 | If V1 = 0, assemble the next card if V2 $\neq$ V3 |
| | | | If V1 $\neq$ 0, assemble the next card if V2 = V3 |
| | IFZ | V1,V2 | Assemble the following V2 cards if the value of V1 = 0. |
| | IFN | V1,V2 | Assemble the following V2 cards if the value of V1 $\neq$ 0. |
| | REPLACE | V1,V2 | Replace cards from alter nol V1 to V2. If V2 is omitted, replace card V1. |
| | DELETE | V1,V2 | Delete cards from alter no. V1 to V2. If V2 is omitted, delete card V1. |
| | INSERT | V1 | Insert following source cards after alter no. V1. |
| | COSY | | Indicates end of modifications and start of the cosy deck. |
| LOC | EQU | V1 | Assign the value V1 to LOC |
| LOC | CON | V1,V2 ... | Assign the values V1, V2, ... to LOC, LOC + 1, ... |
| LOC | BSS | V1 | Assign a block of V1 core cells to LOC |
| LOC | BSSZ | V1 | Assign a block of V1 core cells to LOC |
| LOC | BCD | nnCOMMENT | Assemble the nn following characters in BCD or DPC (nn must be 2 decimal digits). |
| or | | or | |
| LOC | DPC | *COMMENT* | Assemble the characters enclosed with * in BCD or DPC. |
| LOC | VFD | | The VFD card generates a 60 bit word. Field specifications are: |
| | | | Dnn/V1 generate nn bits of display code (nn must be a multiple of 6, the first char must be alphabetic). |

Page 8

Nnn/V1 generate nn bits as an integer. V1 must not be an expression

Ann/V1 generate nn bits as an address (if V1 is relocatable, nn must = 18 and the 18 bit byte must be positioned in bits 0 - 17, 15 - 32, or 30 - 47). V1 must not be an expression

The sum of all nn's must be = 60. If less than 60, the result will be left justified with 0 fill. Examples:

VFD D30/INPUT, N12/0, A18/NAME

END V1 Indicates last card of assembly. If the variable field is non-blank, a main program is assumed and relocation bits will not be punched in the Chippewa binary deck

The following pseudo-ops are available in ASCENT but are meaningful only if relocatable binary decks are punched.

EXT V1,V2, ... Defines the symbols V1,V2 ... as external symbols

ENTRY V1,V2, ... Defines the symbols V1,V2, ... to be entry points

LOC COMMON V1 (L1), V2 (L2), ... Defines common block LOC and arrays V1,V2 ... of lengths L1, L2, ... Blank common is defined by leaving the location field blank. A simple variable must be defined as X (1) and a multiply dimension array as a one dimension array.

## IV. ASPER PSEUDO-OPS

| | | |
|---|---|---|
| ASPER | | Sets assembly mode to Asper |
| ORG | V1 | Sets location counter to the value of V1 |
| ORGR | V1 | Sets location counter to the value of V1 |

All other pseudo-ops are the same as for Ascent except

| | |
|---|---|
| VFD | is not allowed |
| EXT | is not allowed |
| ENTRY | is not allowed |
| COMMON | is not allowed |

## V. ERROR FLAGS

The left margin of the listing may have error flags as follows:

| | |
|---|---|
| O | op-code error. |
| U | undefined symbol in the variable field. |
| D | doubly defined symbol in the variable field or location field. |
| V | VFD error. |
| R | range error for Asper jump instructions. |
| F | field error in the variable field of a CON, BCD or DPC card. |
| L | location field error. |

## VI. FORCING COMMANDS

The instruction following a RJ, JP or PS will be forced upper. A + in the location field will cause the instruction to be forced upper. A - in the location field will cause the instruction to be assembled in the next available portion of the 60 bit word regardless of the preceding instruction.

## VII. SCAN RULES

A card may have information from column 1 through column 72. Either a C in column 1 or a period in column 2 will indicate that this is a remarks card. A period on or after column 11 sets off the remainder of the card as a remark. This is not true on a CON pseudo-op card or in a literal where a period is a decimal point.

The label field may start anywhere from column 2 through column 5. The label ends in column 9.

Column 10 is ignored.

The Op code field may start on or after column 11. The op code is terminated by a field separator. A field separator is one of the set: blank, comma and equal.

The variable field may be any number of field separators from the op code. The scan will attempt to locate the beginning of the variable field up to column 72. However, field separators after the beginning of the variable field are used to delineate operands. Two consecutive field separators will terminate the scan and remarks may follow without any preceding period. The operand expressions in the variable field are evaluated in a left to right scan. Therefore

B26*5+A21 yields (B26 *5) + A21

and A21+B26*5 yields (A21 + B26) * 5.

The use of parentheses is limited to literals and complex constants.

On a CON card the operand entries may be separated by either blanks or commas, but two consecutive commas does not define a zero word. E.G.: CON 8.263E+5,26,27bA53*25bbCONSTANTS FOR J5 defines 4 words, a floating point number and 3 integers.

The numbers generated by literals will be listed in the order of occurrence at the end of the program.

A dollar sign will not define the beginning of the op code field of another instruction.

## VIII. PROGRAMMER DEFINED MACROS

Programmer-defined macros are those which the programmer defines within an ASCENT or ASPER subprogram with a MACRO pseudo instruction. The form of the definition is:

| LOCATION | OPCODE | ADDRESS |
|---|---|---|
| Blank | MACRO | Symbol, list |

where MACRO is the pseudo op code

Symbol is the macro name

List is a sequence of symbols and/or registers separated by commas which define the formal parameters of the macro.

Macros are used, or called, by writing:

The name of the macro in the opcode field; the quantities to be substituted for the dummy parameters in the definition in the address field.

The following rules apply to ASCENT and ASPER use of macros:

1. The definition of a macro must precede the first executable instruction of the subprogram in which it is used.
2. Programmer-defined macros are local to the routine in which the definition appears.
3. A maximum of 100 macros is allowed per subprogram.
4. Macros may be nested to any depth; i.e., macros may be used in the definition of other macros, provided all macros used in definition are themselves defined prior to use. Recursive definition (a macro used in its own definition) is not allowed.
5. Macro names may be any arrangement of letters and numbers which starts with a letter and contains no more than 8 characters.

6. The macro name must not be identical to a machine mnemonic code, a pseudo code, a system macro code, or any other programmer-defined macro in the same routine.
7. A maximum of 16 parameters are allowed in a macro parameter list.
8. The order and count must be the same for formal and actual parameters.
9. In ASCENT subprograms register names and operands in the formal parameter list may be changed by an actual parameter. For example, a parameter Bk may be changed to Ak or to an operand.
10. A zero actual parameter will cause insertion of a zero in the generated instruction if the formal parameter is in the address field, or a blank if the formal parameter is in the location field.
11. A symbol in the location field of a macro call will be assigned to the first word of the macro, and will override any symbol placed there as a parameter.
12. ENDM pseudo-op must be the last instruction in the macro definition.

ASCENT EXAMPLE:

| LOCATION | OPCODE | ADDRESS |
|---|---|---|
| | MACRO | ABC,D,B2,A2,BN, RESULT, X |
| D | SA1 | B2 |
| | SA2 | BN+X |
| | FX6 | X1*X2 |
| | SA6 | RESULT |
| | ENDM | |
| | MACRO | DEF,X4,AM,F,H,Z,L |
| | SAM | OP |

| LOCATION | OPCODE | ADDRESS |
|---|---|---|
| K | ABC | E,B3,A3,Z,F,0 |
| H | FX7 | X6/X4 |
| L | SA7 | G |
| | ENDM | |

Using the definitions above, a macro call of

| | | |
|---|---|---|
| | DEF | X5,A5,LOC1,LOC2,U*V+Q-10,0 |

would generate the following set of instructions:

| | | |
|---|---|---|
| | SA5 | OP |
| K | SA1 | B3 |
| | SA3 | U*V+W-12B+OB |
| | FX6 | X1*X2 |
| | SA6 | LOC1 |
| LOC2 | FX7 | X6/X5 |
| SA7 | G | |

ASPER EXAMPLE:

| LOCATION | OPCODE | ADDRESS |
|---|---|---|
| | MACRO | XYZ,OP,A,B,C |
| | LDM | A,B |
| | OP | C |
| | STM | A,B |
| | ENDM | |

Using the definition above, a macro call of

| | | |
|---|---|---|
| LOC | XYZ | SBD,D1,D2,D3 |

would generate the following set of instructions:

| | | |
|---|---|---|
| LOC | LDM | D1,D2 |
| | SBD | D3 |
| | STM | D1,D2 |

## IX. RELOCATION RULES FOR SUBROUTINES

A symbol is any arrangement of letters and numbers which starts with a letter and contains up to 8 characters. A symbol is relocatable if it occurs in the label field of an instruction or pseudo-operation that defines a core location. A symbol is non-relocatable if it occurred on the label field of an EQU card whose operand is an integer. Operands that consist of expressions of non-relocatable symbols will not be relocated. Operands that consist of expressions that are mixtures of relocatable and non-relocatable symbols will not be relocated if (a) a relocatable symbol is the operand in a multiply (*) or divide (/) operation; (b) an expression consists of the sum or difference of 2 or more relocatable symbols.

In the Chippewa system relocation bits are bits 16 and 17 of a 30 bit instruction. Bit 16 signifies common. Bit 17 signifies an ordinary symbol. Both 16 and 17 on or off signify a constant. Certain valid instructions although handled properly by the assembler may be loaded improperly.

E.G.: SX7 220314B. This instruction has bit 16 set and will be relocated in common.

The first two words of a subroutine define its name, length, and relocatable length for the loader. The required words may be generated by VFD pseudo-ops. They have the form:

```
VFD D24/NAME, N18/0, A18/END

VFD A18/RELOC, A18/END, N24/PARAMS
```

The symbol END should point to the last core cell used by a routine. The symbol RELOC should point to the first constant used by a routine. Both END and RELOC must be non-relocatable. To achieve this they should be defined in terms of relocatable symbols that have been multiplied by the integer, 1. A subroutine should therefore have the following format:

IX. (Continued) Page 2

```
        ENTRY   NAME
        VFD     D24/NAME, N18/0, A18/END
        VFD     A18/RELOC, A18/END, N24/N
        BSS     N    N= no. of arguments to subroutine
NAME    CON     0
        ---
        (SUBROUTINE INSTRUCTIONS)
        ---
RELOC   EQU     **1+1  start of constants
        ---
        (SUBROUTINE CONSTANTS)
END     EQU     **1+1  end of constants
        END
```

## X. ADDITIONAL NOTES

1. The peripheral routines 2RC and 2PC have been modified for use withASCENT. 2RC will input the entire 80 columns of a binary card if col 1. = 0005. 2PC will punch an 80 column card image if the file name is P80C. The ASCENT assembler requires a field length of $35000_8$.
2. PAS decks may be translated to ASPER by setting all $20_8$ non-zero.
3. CLAS decks may be translated to ASCENT by setting all $17_8$ non-zero.
4. A PS instruction generates 30 bits of zeros.
5. A JP instruction generates an 02 op code.

### TABLE SIZES

The tables have been defined on EQU cards and their lengths are easily changed by re-assembly. Ascent tables have the following lengths:

| SYMBOL | LENGTH | DEFINITION |
|---|---|---|
| AS25 | 1000 | max number of symbols |
| AS32 | 200 | max INSERT, DELETE, REPLACE cards |
| AS33 | 500*10 | insert table size |
| AS78 | 25 | number of literals |
| AS90 | 50 | number of macro names |
| AS91 | 400 | macro skeleton table length |

CONTROL DATA CORPORATION

Development Division - Applications

# THE DEAD START PROCESS AND THE SYSTEM LOADER

Chippewa Operating System

# THE DEAD START PROCESS AND THE SYSTEM LOADER

## INTRODUCTION

The dead start process requires that a short program (up to 12 instructions) be set up on the matrix of toggle switches on the dead start panel. When the dead start switch is toggled, this dead start program is transmitted to peripheral processor zero's memory and executed. The dead start program in turn transmits a bootstrap program to another peripheral processor. This bootstrap program brings in the system loader from the library tape and transfers control to it. The system loader transfers a resident program to each peripheral processor, causes the Display and Monitor programs to be loaded, loads the central memory resident, library, and tables, and places the remaining library programs on the disk. It then inititates execution of the Display and Monitor programs.

## THE IAM INSTRUCTION

A detailed understanding of the dead start loading process requires some familiarity with the functioning of the IAM instruction. The IAM instruction is a 24-bit instruction: the d portion of the instruction holds the channel number and the m portion of the instruction contains the address in peripheral processor memory where the first data word is to be stored. The A register is assumed to contain the number of words to be read. The functioning of the IAM instruction is shown in figure 1. Note the following points:

- During execution of the IAM instruction, the contents of the P register are stored in location 0, and the P register used to hold the memory address for the next word to be stored. At the time the contents of the P register are stored, P holds the address of the second word (m

portion) of the IAM instruction. Before exiting the instruction, the contents of location 0 are read, incremented by one, and placed in the P register to provide the address of the next instruction.

- The IAM instruction tests the word count in the A register to see if it has been reduced to one: if so, (A) is reduced by one and the instruction exited. Therefore, if the IAM instruction is entered with the contents of the A register equal to zero, the word count is effectively $77777_8$.
- The IAM instruction may be exited in one of two ways: (1) because the word count has been reduced to zero or (2) because the channel has become inactive. If the word count has not been reduced to zero and the channel is active, exit will not take place even though no data is being read: the processor will idle in trip 4, waiting for the channel to become full.

## THE DEAD START SEQUENCE

When the dead start switch is toggled, the following sequence is initiated:

- The Master Clear signal is generated
- The A register of each peripheral processor is set to $10000_8$: the P register of each peripheral processor is set to zero
- The K register of each peripheral processor is set to 712 (trip 4 of an IAM instruction)
- All channels are set to empty and active
- All peripheral processors are connected to their respective channels (i.e., PP0 to channel 0, PP1 to channel 1, etc.) by setting the appropriate channel number in each processor's Q register
- The first synchronizer on each channel is selected: the first unit on that synchronizer is selected
- The dead start synchronizer is selected on channel 0
- The program on the dead start panel is transferred to PP0 memory: first, a zero byte is transmitted (stored in location 0); next, the 12 bytes from the panel switches are transmitted (stored in location 1 - 14); finally, another zero byte is transmitted (stored in location 15)

# THE IAM INSTRUCTION

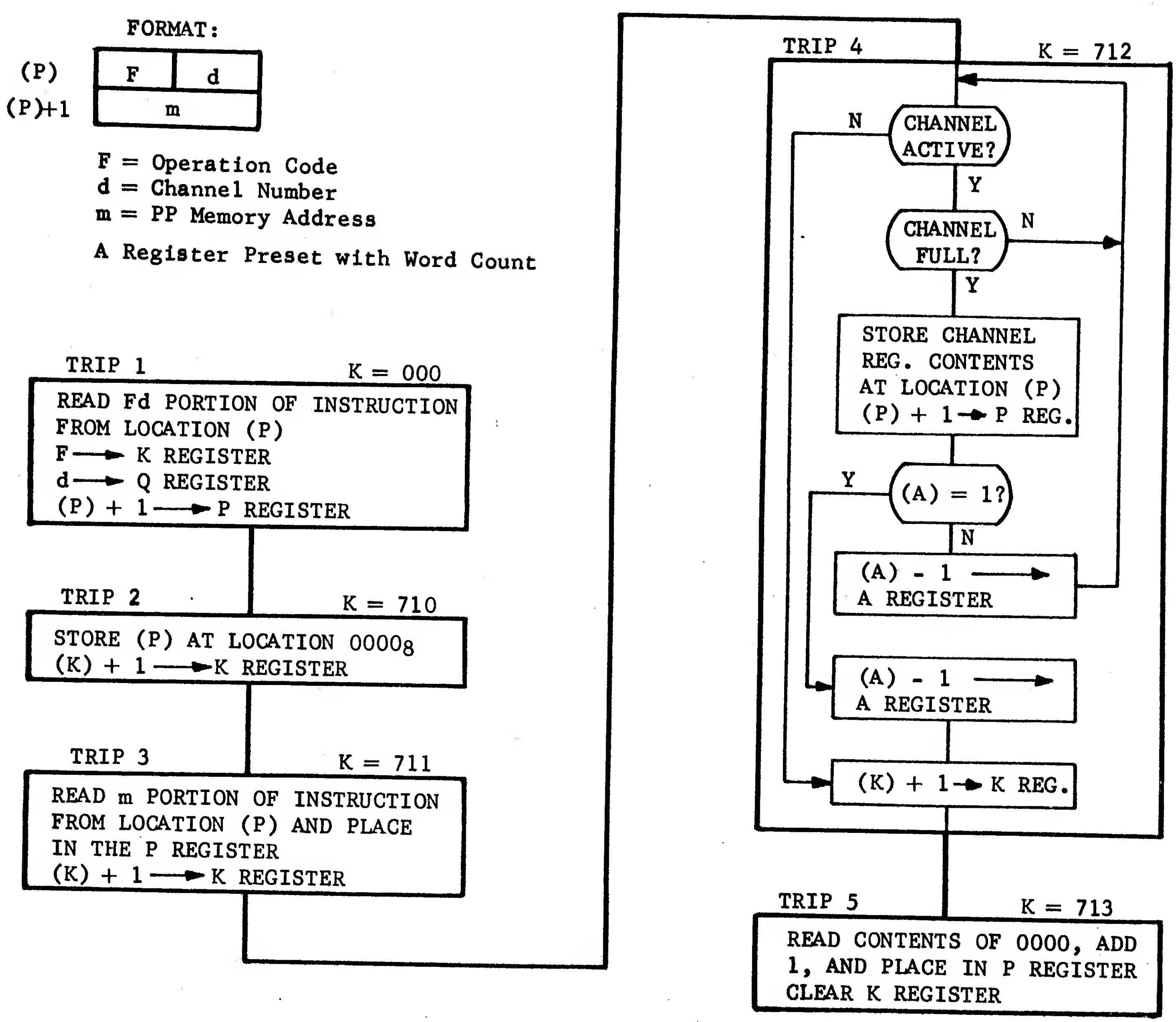



Figure 1

- The dead start synchronizer disconnects channel 0, initiating the execution of the dead start program

Peripheral processor zero treats the data sent by the dead start synchronizer as it would data arriving from any other controller. When the dead start synchronizer disconnects from channel zero, peripheral processor zero exits from the IAM instruction. In exiting, the contents of location 0 are incremented by 1 and used as the address of the next instruction. Since this location was cleared to 0 by the dead start process, the address of the next instruction is 0001: this location holds the first instruction of the program sent by the dead start synchronizer from the dead start panel.

## THE DEAD START PROGRAM

The dead start program is shown in figure 2. The purpose of the dead start program is to transmit a bootstrap program to peripheral processor xx (PPxx), where xx is the channel number of the controller on which the system tape is mounted. The dead start program begins by transmitting a block of 8 words on channel xx. PPxx is connected to this channel and is idling in trip 4 of an IAM instruction: it will therefore read in these 8 words and store them in its memory beginning at location 0. PPxx will not, however, begin execution yet, since the channel is still active and the word count has not been reduced to zero.

The dead start program next disconnects channel xx: when channel xx becomes inactive, PPxx exits from the IAM instruction and begins execution of the bootstrap program. In exiting from the IAM instruction, the contents of location 0 are read, incremented by 1, and used as the address of the next instruction. Since the first word of the 8-word block sent by PP0 was equal to zero, this address is equal to 0001, and the instruction at this address is read and executed.

The dead start program then issues an input instruction for channel 13: since

## THE DEAD START PROGRAM

Good controllers only

xx = CHANNEL NUMBER FOR CONTROLLER ON WHICH SYSTEM TAPE IS MOUNTED

| | PPO MEMORY LOCATION | INSTRUCTION | FUNCTION |
|---|---|---|---|
| THESE INSTRUCTIONS ARE EXECUTED IN PPO | 0001 | 1410 | SET THE A REGISTER TO $10_8$ |
| | 0002 | 73xx | OUTPUT 8 WORDS BEGINNING AT LOCATION 0006 ON CHANNEL xx (PROCESSOR xx WILL STORE THESE INTO ITS MEMORY BEGINNING AT LOCATION 0) |
| | 0003 | 0006 | |
| | 0004 | 75xx | DISCONNECT CHANNEL xx (THIS PERMITS PROCESSOR xx TO EXIT FROM THE IAM INSTRUCTION AND BEGIN EXECUTION OF THE PROGRAM IT HAS RECEIVED) |
| | 0005 | 7113 | SET PPO TO INPUT FROM CHANNEL 13 (CHANNEL 13 IS EMPTY AND ACTIVE: PPO WILL IDLE IN THIS INSTRUCTION UNTIL CHANNEL 13 BECOMES FULL) |
| | 0006 | 0000 | |
| THESE INSTRUCTIONS ARE TRANSFERRED TO AND EXECUTED IN PPxx | 0007 | 77xx | |
| | 0010 | 2000 | |
| | 0011 | 77xx | |
| | 0012 | 2020 | |
| | 0013 | 74xx | |
| | 0014 | 71xx | |
| | 0015 | 0000 | CLEARED DURING DEAD START |

See note)

Figure 2

channel 13 is empty and active, PP0 will idle in trip 4 of this instruction waiting for channel 13 to become full.

## THE LOADER BOOTSTRAP

The bootstrap program (figure 3) in PPxx issues the necessary function, activate, and input instructions to read the first record on the system tape into its memory beginning at location 0. When this record, which contains the loader program, has been read, PPxx will exit the IAM instruction when the controller disconnects the channel upon detecting the end-of-record gap. PPxx, in exiting the IAM instruction, reads the contents of location 0, adds 1 to it, and uses this as the address of the next instruction. Location 0 contains the first word of the record read from tape: thus, this word supplies the address of the first instruction of the loader program.

## THE LOADER PROGRAM

The layout of the loader program in PPxx is shown in figure 4. As mentioned earlier, the first word of the loader program (location 0) contains the address - 1 of the first instruction of the loader. The loader program also contains a peripheral processor resident package in locations 0001 - $0777_8$. This package is transmitted by the loader to each of the other peripheral processors. The resident program is contained in locations 0100 - $0777_8$: locations 75, 76, and $77_8$ contain the values 60, 61, and 62, respectively. These values are the central memory addresses of the Input Register, the Output Register, and the Message Buffer for PP1, and must be modified when the resident package is transmitted to processors other than PP1.

At the time the loader program begins execution, all channels except the channel corresponding to the processor containing the loader program (PPxx) are active and empty: their corresponding processors are idling in an IAM instruction, waiting for input. Channel xx, however, was disconnected by the tape controller

## THE DEAD START PROGRAM

### LOADER BOOTSTRAP

| PPxx MEMORY LOCATION | INSTRUCTION | FUNCTION |
|---|---|---|
| 0000 | 0000 | NOT EXECUTED (ADDRESS - 1 OF FIRST INSTRUCTION) |
| 0001<br>0002 | 77xx<br>2000 | ISSUE FUNCTION CODE: SELECT UNIT 0 |
| 0003<br>0004 | 77xx<br>2020 | ISSUE FUNCTION CODE: SELECT BINARY READ |
| 0005 | 74xx | ACTIVATE CHANNEL xx (xx DISCONNECTED BY PPO PROGRAM) |
| 0006<br>0007 | 71xx<br>0000 | INPUT (A) WORDS FROM CHANNEL xx BEGINNING AT LOCATION 0000 (THE A REGISTER HAS NOT BEEN USED AND STILL CONTAINS 0) |

- THE TAPE CONTROLLER WILL DISCONNECT THE CHANNEL WHEN THE END-OF-RECORD GAP IS DETECTED AND THUS CAUSE THE PROCESSOR TO EXIT FROM THE 71 INSTRUCTION.
- THE FIRST WORD FROM TAPE WILL BE READ INTO LOCATION 0: ON EXIT FROM THE IAM INSTRUCTION, THE CONTENTS OF LOCATION 0 ARE READ, INCREMENTED BY 1, AND USED AS THE ADDRESS OF THE NEXT INSTRUCTION.

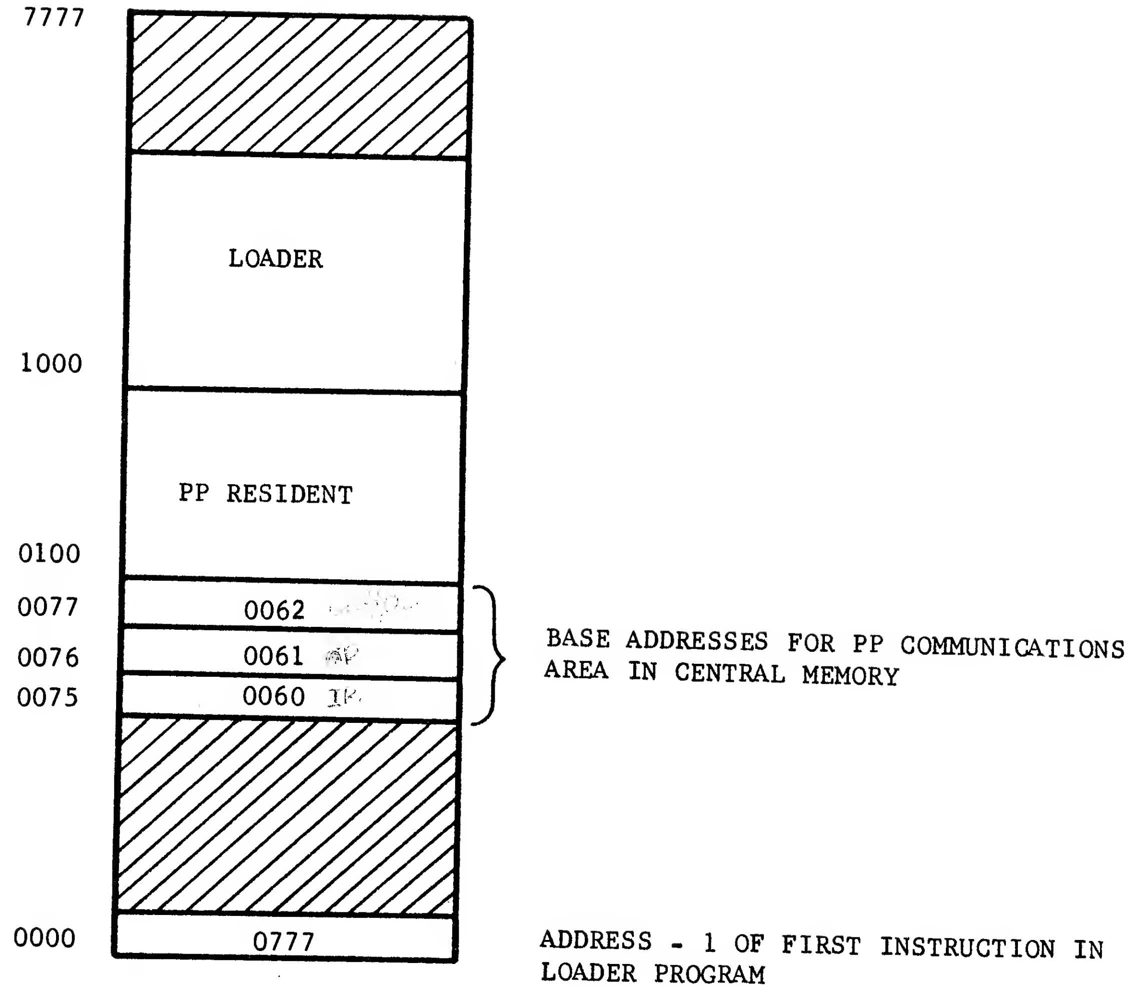
Figure 3

when the end-of-record gap following the loader program was detected, and is therefore inactive. The loader program searches for an inactive channel in order to determine which processor it resides in: it also inserts this channel number in the appropriate I/O instructions.

The loader program then proceeds to determine if the system tape is mounted on a 607-B unit or a 626-B unit, and modifies the function codes accordingly. Transfer of the resident package to each of the other processors then takes place. The loader first outputs a single word to the receiving processor, which stores it in its memory at location zero. Since the receiving processor is in trip 4 of an IAM instruction, it will, upon exiting this instruction, use the contents of location 0 as the address - 1 of the next instruction it is to execute. For processors 1 - 8, this address is $77_8$: the address - 1 of the first instruction of the resident program. For processors 0 and 9, this address is $777_8$: the address - 1 of the first instruction of the MTR and DSD programs, respectively. After transmitting this single word, the loader then transmits the resident package, which the receiving processor stores in its memory beginning at location 0001. The receiving processor does not exit the IAM instruction at this time, however, since the conditions for exiting (either word count reduced to zero or channel inactive) have not been met. As the transfer of each resident takes place, the loader program modifies the Input Register, Output Register, and Message Buffer pointers to the proper values for each processor.

When all processors have been loaded with the resident package, the loader program then proceeds to load the MTR and DSD programs from the system tape into processors 0 and 9, respectively.

The format of the system tape is illustrated in figure 5. The tape contains a single file of binary records: a full physical record contains $1000_8$ CM words. A logical record, such as the MTR program or the CM resident, may be composed of more than one physical record: the last physical record for a specific program may be a short record of less than $1000_8$ CM words. The end of a logical record is indicated when a short physical record is processed or when a zero length record




<u>LOADER PROGRAM LAYOUT IN PPxx</u>

(RECORD 1 ON SYSTEM TAPE)

**Figure 4**

(4 PP words) is detected, except for the disk library routines: the end of each disk library routine is indicated by a short record only. The end of a library is indicated by a zero length record.

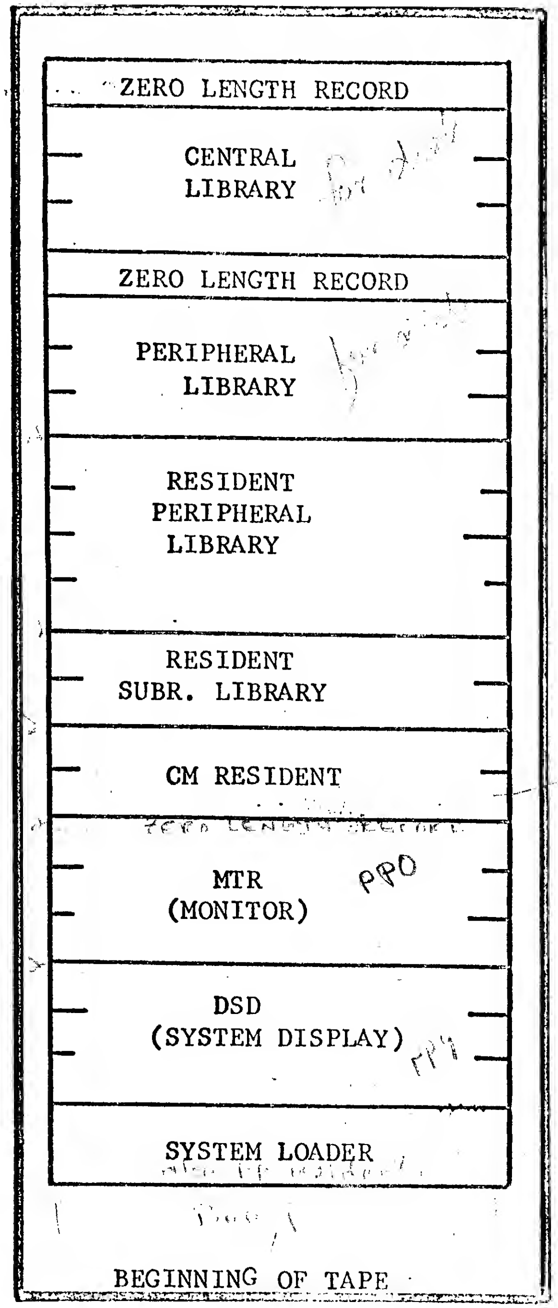
The loader reads the records comprising the DSD program, transferring each record as it is read to PP9: when a short record is processed or a zero length record is detected, loading of the DSD program is complete. This process is repeated for the MTR program records.

The CM Resident is loaded next. This resident contains table pointers and initial values for certain tables, such as the track reservation tables. The resident subroutine library is loaded using the RSL pointer from the CM Resident to provide the starting address: the resident peripheral library (RPL) is similiarly loaded. In all three cases (CM Resident, RSL, and RPL), records are read and transferred to central memory until either a zero length record is detected or a short record is processed.

The loader program then disconnects the channels for each of the other processors, permitting these processors to exit from the IAM instruction and begin execution of their programs. Now that MTR is executing, the loader program can utilize the assistance of MTR in loading the libraries on the disk.

The loader requests a track from MTR via its resident, and picks up the Peripheral Library Directory pointer from central memory in order to obtain the starting address of the directory. It then reads a record from the system tape, builds the PLD entry and writes it in the directory, and transfers the record to the disk. The next record is then read from tape and written to the disk: this process continues until a short record is processed, indicating that a complete program has been transferred. The next record is read from tape, the directory entry constructed and written in the directory, and the process of reading records from tape and transferring them to the disk repeated. The end of a library is indicated by the detection of a zero length record.

When the peripheral library has been transferred to the disk, the transfer of the central library to the disk is initiated and executed in the same manner.




lowest 17000 of cm

- SYSTEM TAPE CONTAINS A SINGLE FILE OF BINARY RECORDS
- A FULL PHYSICAL RECORD IS A BLOCK OF $1000_8$ CM WORDS takes 10000 frames
- A LOGICAL RECORD (i.e., MTR, RPL, etc.) MAY BE COMPOSED OF A NUMBER OF PHYSICAL RECORDS
- THE END OF A LOGICAL RECORD IS INDICATED WHEN A SHORT RECORD IS PROCESSED OR WHEN A ZERO LENGTH RECORD (4 BYTES) IS DETECTED
- THE END OF PERIPHERAL LIBRARY OR CENTRAL LIBRARY ROUTINES IS INDICATED BY A SHORT RECORD ONLY: THE END OF THE LIBRARY IS INDICATED BY A ZERO LENGTH RECORD

## *SYSTEM TAPE FORMAT*

Figure 5

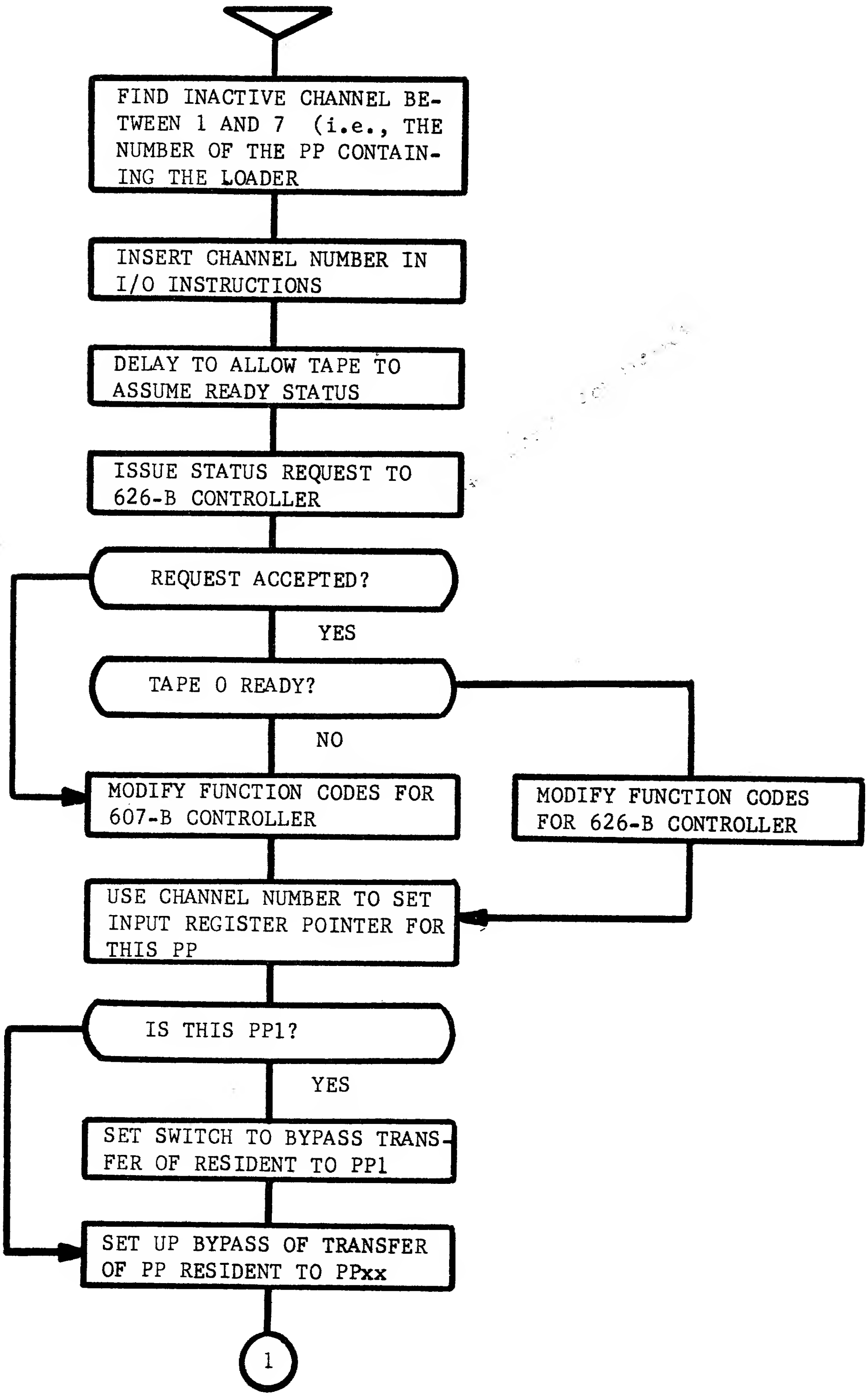
When a half track is filled during library transfers to the disk, the loader program requests a new half track from MTR via its (the loader's) resident program. The resident's POSITION DISK routine is used to position the disk to the new half track position.

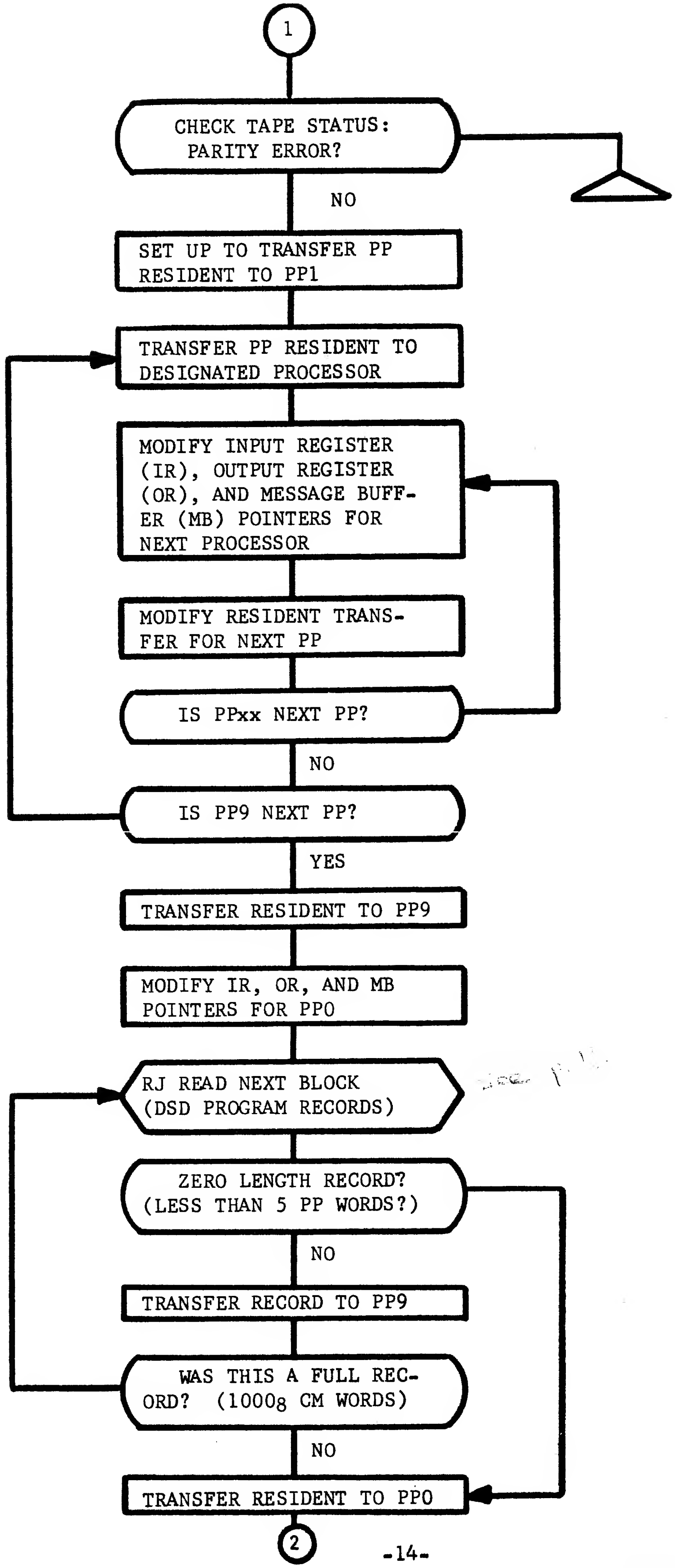
When the transfer of the central library to the disk is completed, the loader program exits to the idle loop of its resident.

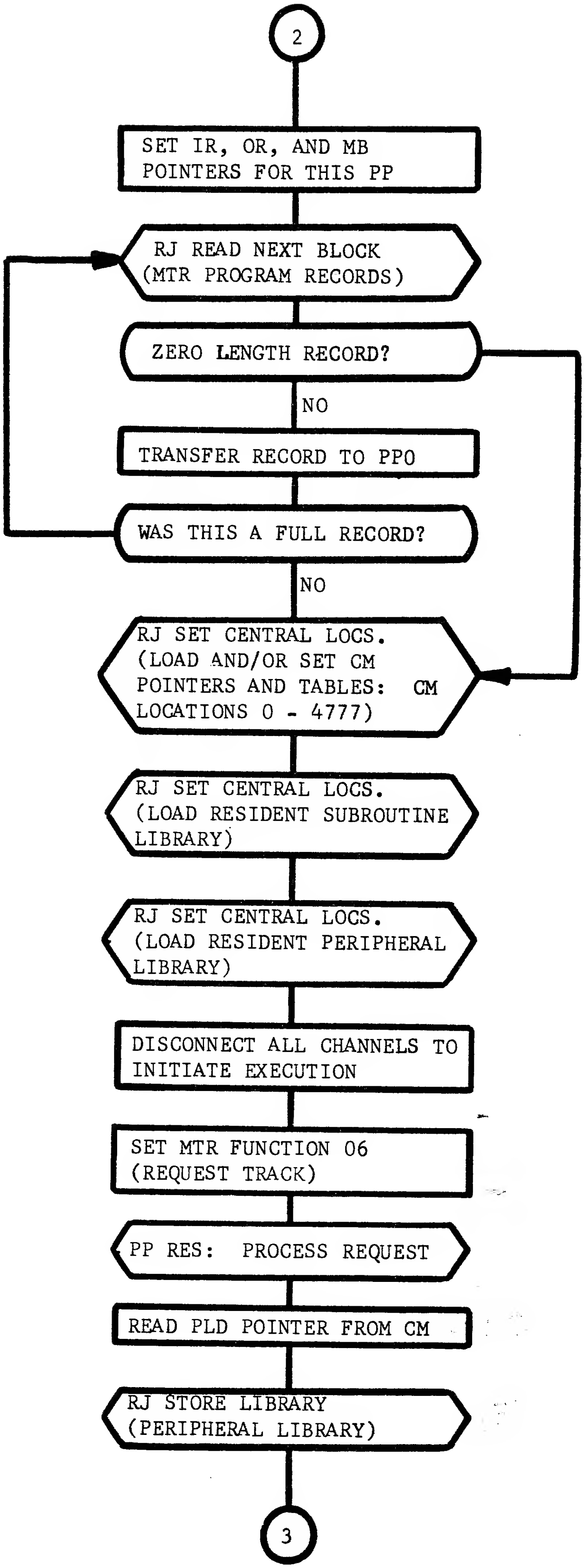
Notes:

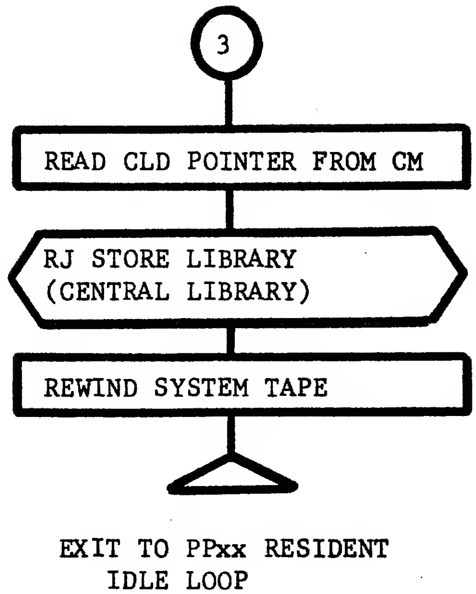
1. For the loading process described, the system tape should be mounted on unit zero of the first controller on channel xx. If channel xx has both 607-B and 626-B controllers, the unused controller's unit zero should be made not ready. The channel xx may be any channel from 1 to 9.
2. In addition to the bootstrap routine described, a variety of others are in use. Many of these use a one-card loader.

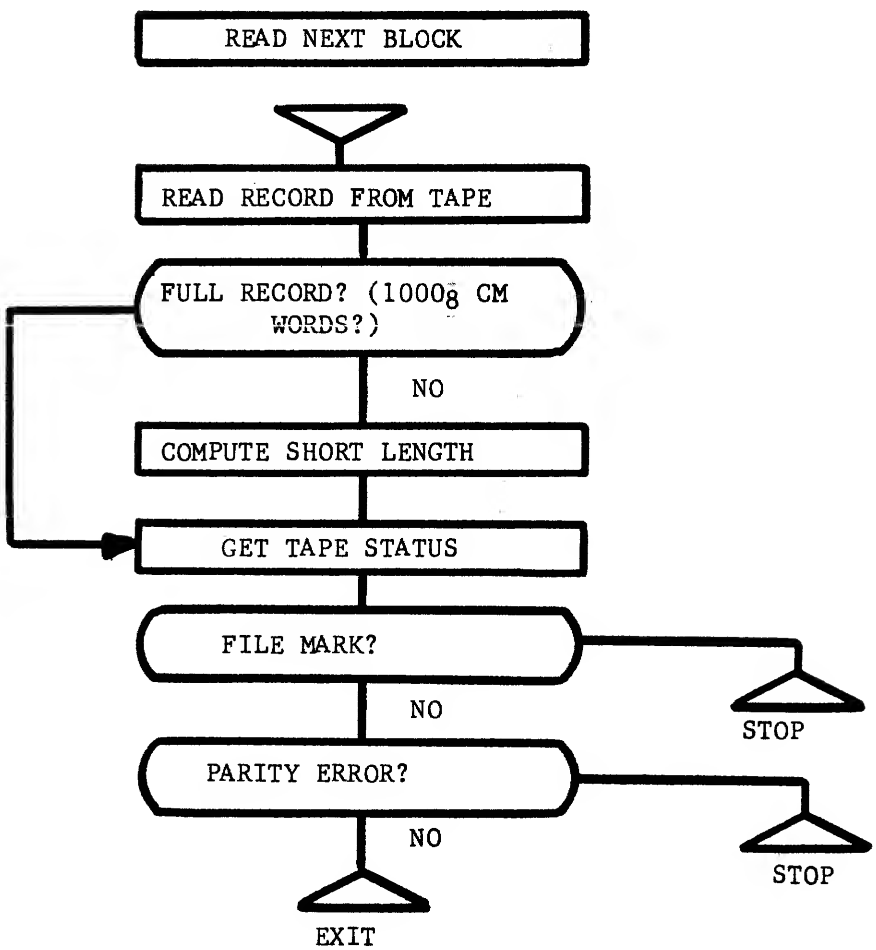
SYSTEM TAPE LOADER



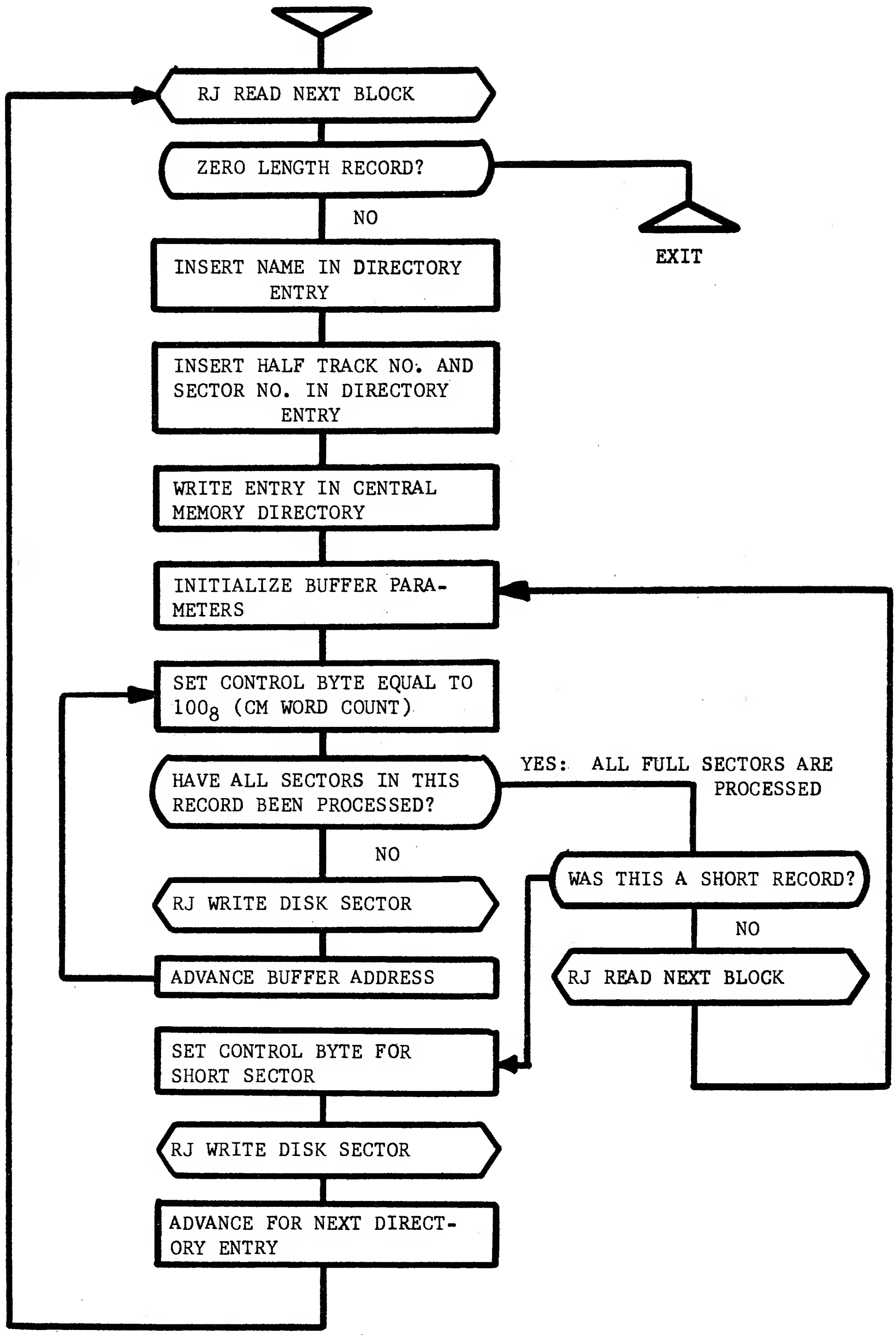

1
CHECK TAPE STATUS: PARITY ERROR?
NO
SET UP TO TRANSFER PP RESIDENT TO PP1
TRANSFER PP RESIDENT TO DESIGNATED PROCESSOR
MODIFY INPUT REGISTER (IR), OUTPUT REGISTER (OR), AND MESSAGE BUFFER (MB) POINTERS FOR NEXT PROCESSOR
MODIFY RESIDENT TRANSFER FOR NEXT PP
IS PPxx NEXT PP?
NO
IS PP9 NEXT PP?
YES
TRANSFER RESIDENT TO PP9
MODIFY IR, OR, AND MB POINTERS FOR PP0
RJ READ NEXT BLOCK (DSD PROGRAM RECORDS)
ZERO LENGTH RECORD? (LESS THAN 5 PP WORDS?)
NO
TRANSFER RECORD TO PP9
WAS THIS A FULL RECORD? ($1000_8$ CM WORDS)
NO
TRANSFER RESIDENT TO PP0
2

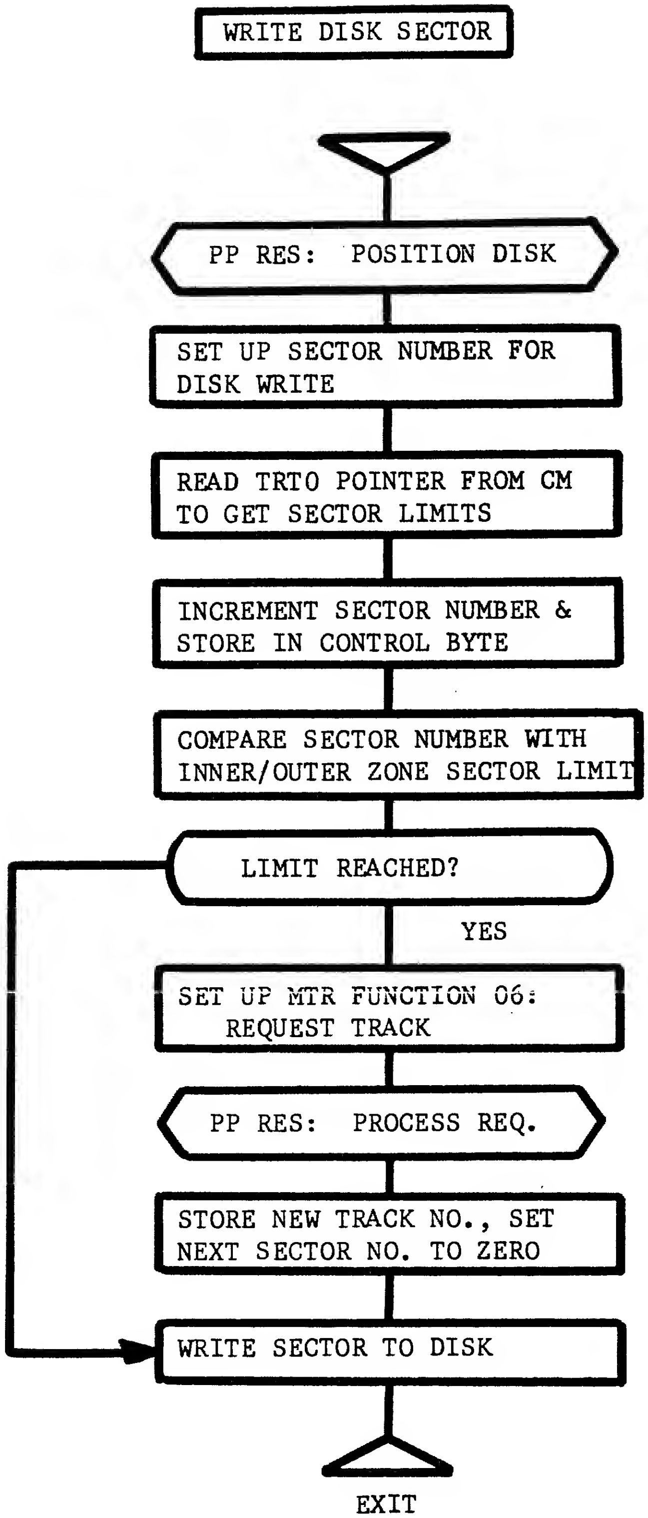

2
SET IR, OR, AND MB POINTERS FOR THIS PP
RJ READ NEXT BLOCK (MTR PROGRAM RECORDS)
ZERO LENGTH RECORD?
NO
TRANSFER RECORD TO PPO
WAS THIS A FULL RECORD?
NO
RJ SET CENTRAL LOCS. (LOAD AND/OR SET CM POINTERS AND TABLES: CM LOCATIONS 0 - 4777)
RJ SET CENTRAL LOCS. (LOAD RESIDENT SUBROUTINE LIBRARY)
RJ SET CENTRAL LOCS. (LOAD RESIDENT PERIPHERAL LIBRARY)
DISCONNECT ALL CHANNELS TO INITIATE EXECUTION
SET MTR FUNCTION 06 (REQUEST TRACK)
PP RES: PROCESS REQUEST
READ PLD POINTER FROM CM
RJ STORE LIBRARY (PERIPHERAL LIBRARY)
3


3
READ CLD POINTER FROM CM
RJ STORE LIBRARY (CENTRAL LIBRARY)
REWIND SYSTEM TAPE
EXIT TO PPxx RESIDENT IDLE LOOP

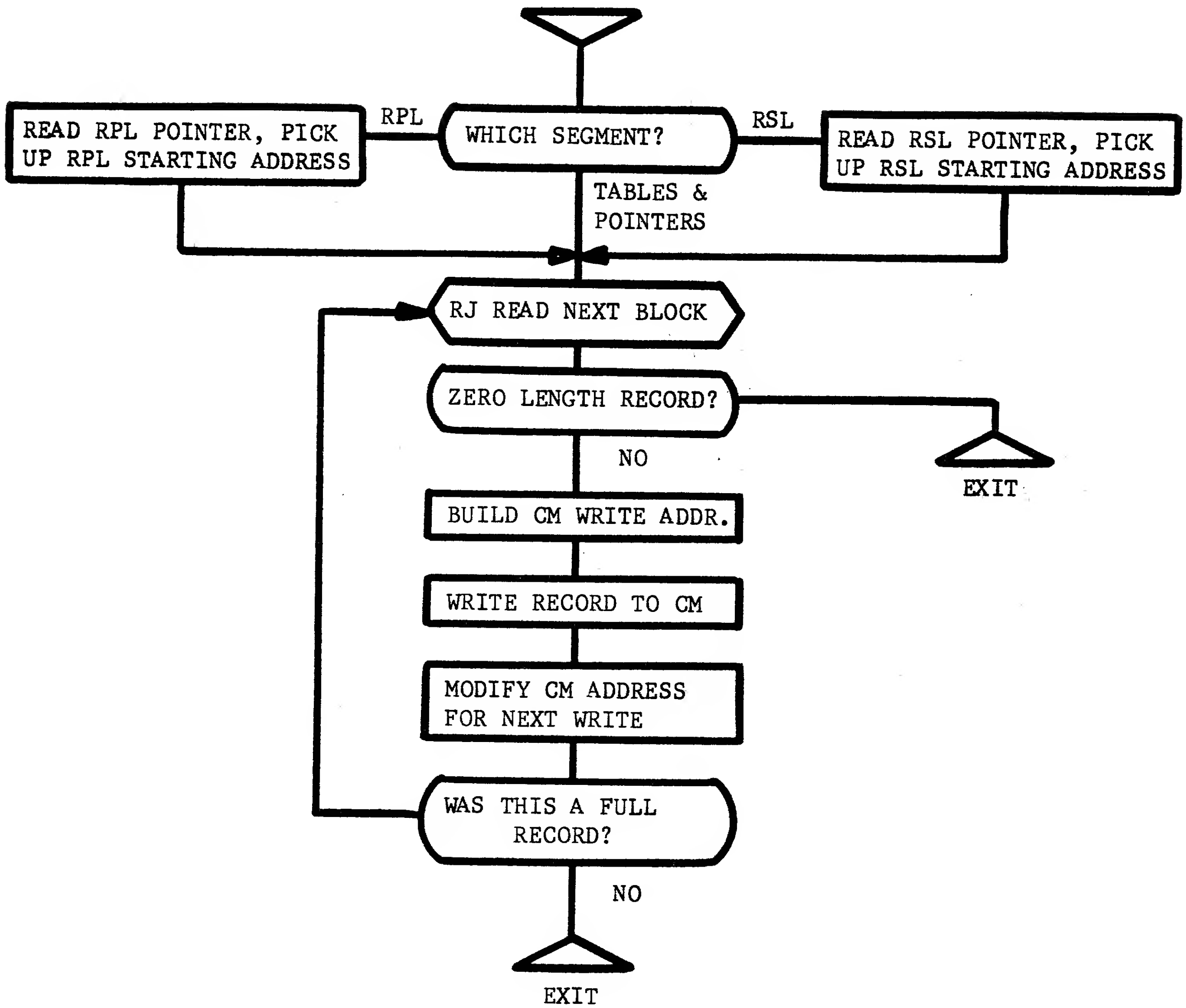

READ NEXT BLOCK
READ RECORD FROM TAPE
FULL RECORD? ($1000_8$ CM WORDS?)
NO
COMPUTE SHORT LENGTH
GET TAPE STATUS
FILE MARK?
STOP
NO
PARITY ERROR?
STOP
NO
EXIT

STORE LIBRARY




WRITE DISK SECTOR
PP RES: POSITION DISK
SET UP SECTOR NUMBER FOR DISK WRITE
READ TRTO POINTER FROM CM TO GET SECTOR LIMITS
INCREMENT SECTOR NUMBER & STORE IN CONTROL BYTE
COMPARE SECTOR NUMBER WITH INNER/OUTER ZONE SECTOR LIMIT
LIMIT REACHED?
YES
SET UP MTR FUNCTION 06: REQUEST TRACK
PP RES: PROCESS REQ.
STORE NEW TRACK NO., SET NEXT SECTOR NO. TO ZERO
WRITE SECTOR TO DISK
EXIT

SET CENTRAL LOCATIONS



CONTROL DATA CORPORATION

Development Division - Applications

# POOL PROCESSORS AND PERIPHERAL PROCESSOR RESIDENTS

Chippewa Operating System

# POOL PROCESSORS AND PERIPHERAL PROCESSOR RESIDENTS

## INTRODUCTION

In the Chippewa Operating System, the System Display program (DSD) and the Monitor program (MTR) permanently reside in two of the ten peripheral processors. MTR and DSD reside in processors 0 and 9, respectively. The remaining processors, 1 - 8, form a pool of processors to which MTR may assign tasks as required. These pool processors have no fixed assignments: any processor may be assigned to the execution of any system routine, and it is possible that more than one processor may be executing the same routine at the same time. All ten processors contain a small resident program which handles the communications between pool processor programs and the Monitor, and initiates the execution of these programs as directed by MTR.

## POOL PROCESSOR STRUCTURE

The structure of a pool processor is illustrated in figure 1. The resident program is contained in locations 0100 - 0772: locations 75, 76, and 77 contain pointers to the Input Register, the Output Register, and Message Buffer in central memory. When directed to do so by MTR, the resident loads a program into its memory and executes it: since that program remains in that processor only for the period of time required to perform its function, it is called a transient program. Transient programs occupy locations 0773 - 1772, although the first instruction is at location 1000. Transient programs generally load overlays to perform specific tasks. For example, CIO, which is a transient program, calls various overlays depending on the task (read, write, backspace) and the equipment (disk, tape, etc.) specified. Overlays are loaded into memory beginning

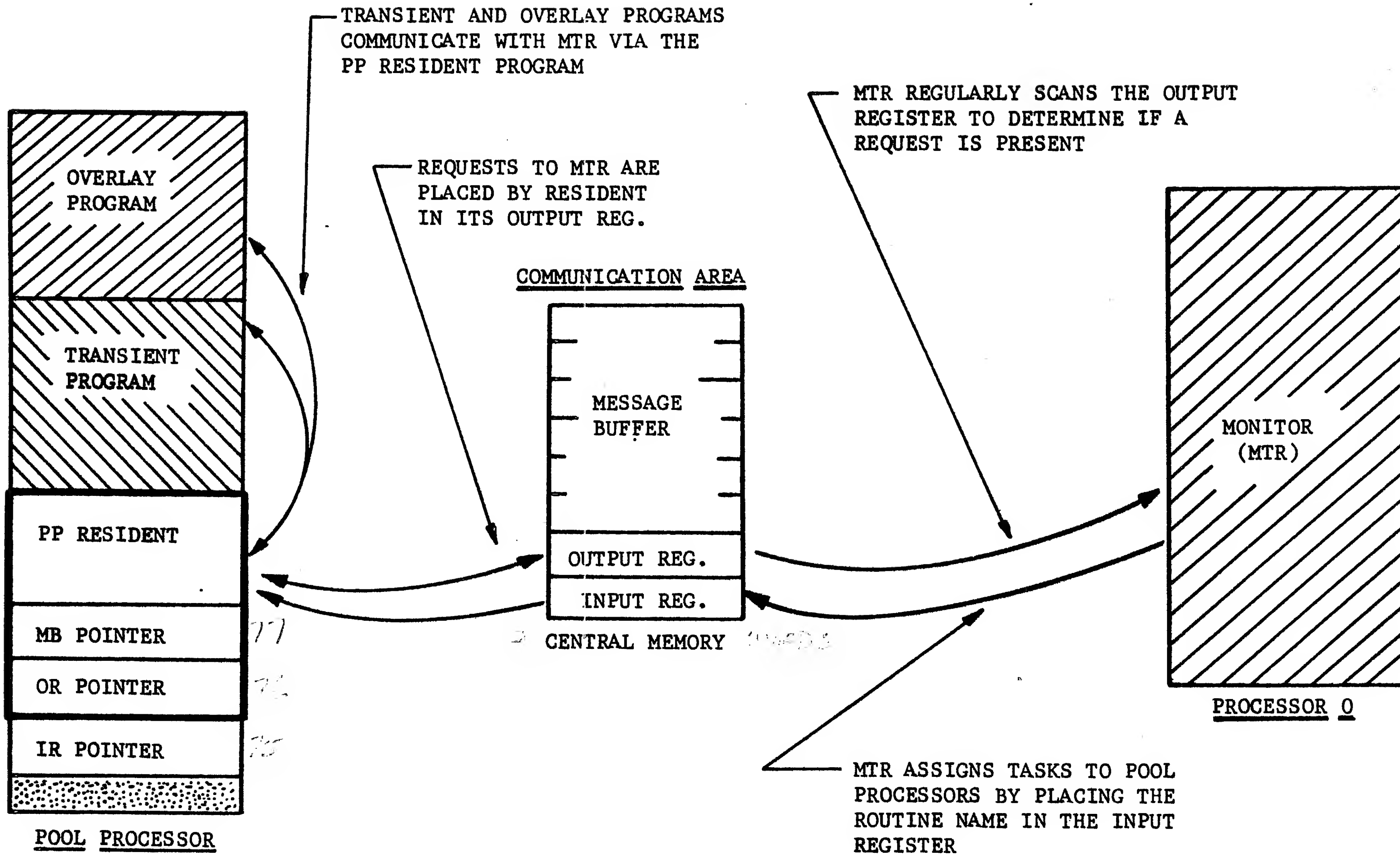



Figure 1

*POOL PROCESSORS & PP RESIDENT*

at location 1773: the first instruction falls at location 2000. Overlays are generally entered via a return jump. Transient programs have names beginning with a letter (CIO, EXU) or the numeral 1 (1BJ, 1LT): overlays have names beginning with the numeral 2 (2WD, 2BP, etc.).

Both transient and overlay programs, as well as the resident program, make extensive use of the low core locations 01 - 74.

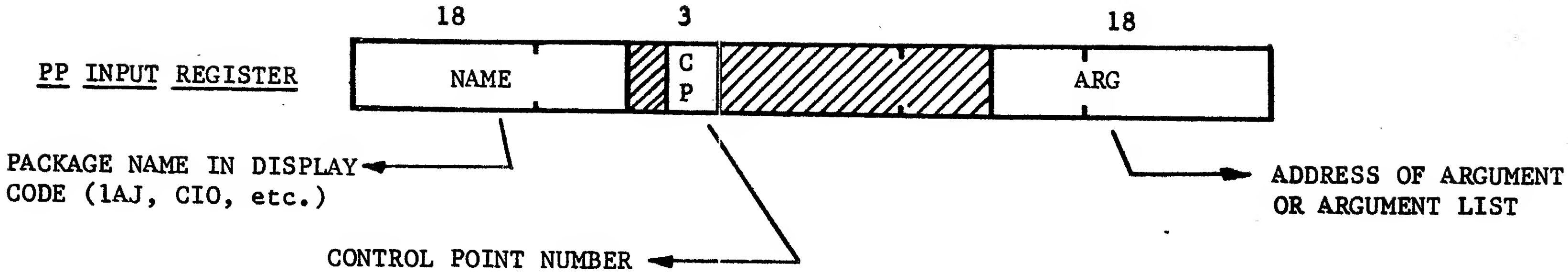
THE RESIDENT

The peripheral processor resident program has two main functions to perform:

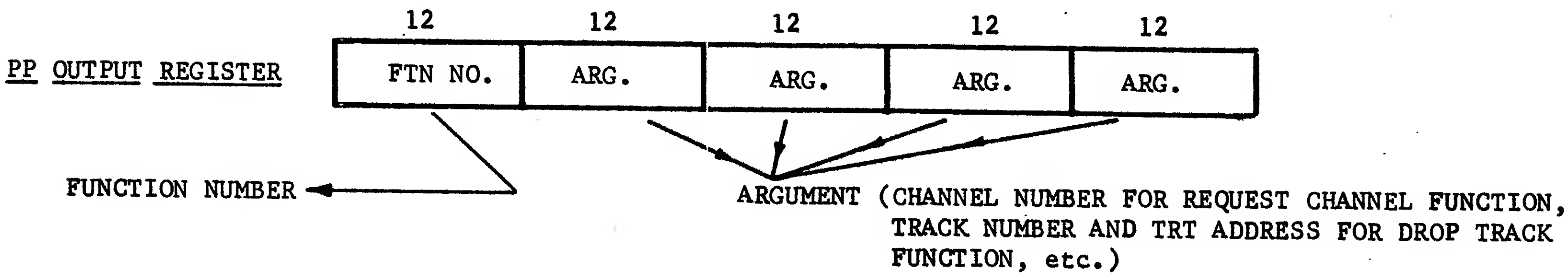
- all communication between MTR and the transient or overlay programs is handled by the resident;
- the resident, when directed by MTR, loads transient programs from either the RPL or the disk library and initiates the execution of these programs.

Communication between MTR and the resident programs is carried out through the use of ten communication areas in central memory, one for each processor. (Note: MTR on occasion communicates with itself by this means.) Each communication area consists of a one-word Input Register, a one-word Output Register, and a six-word Message Buffer. Pool processors address these areas by means of pointers in locations 75 - 77.

MTR assigns a task to a pool processor by placing the request in the processor's Input Register. The format of the request is shown in figure 2. The name of the program package which is to be loaded and executed appears in the high-order 18 bits of the Input Register. This name consists of three display code characters, such as CIO, 1AJ, etc. The number of the control point to which this package is assigned is contained in the low-order 3 bits of byte 2 of the Input Register. The address of the argument(s) required by the package appears in the low-order 18 bits of the Input Register. The request remains in the Input Register until the task is completed. On completion of a task, the transient program requests MTR to release the processor: MTR then clears the processor's




*MTR REQUEST TO RESIDENT*




SOME FUNCTIONS USE THE MESSAGE BUFFER TO SEND ARGUMENTS TO MTR (e.g., DAYFILE MESSAGE FUNCTION)

*RESIDENT REQUEST TO MTR*

Figure 2

Input Register. The Input Register of a pool processor is thus clear only when the processor is idle. When MTR needs a pool processor to assign to a task, it searches the communication areas for a cleared Input Register: when one is found, the corresponding processor is assigned to the task.

All communication between the Monitor and the transient and overlay programs is handled by the resident program. MTR performs a variety of functions, each of which is identified by a function code of one or two octal digits. Some of these functions are listed below:

| Code | Function |
|---|---|
| 01 | Process Dayfile Message |
| 02 | Request Channel |
| 07 | Drop Track |
| 12 | Release PP |
| 33 | Assign Equipment |

To transmit a request to MTR, the resident places the request in its Output Register. The format of this request is illustrated in figure 2. Byte 1 of the Output Register contains the function code in the low-order bit positions. Bytes 2 - 5 are used for arguments: the number of argument bytes depends on the particular function. Thus, for a Request Channel function (function number 2), the channel number is placed in byte 2. For a Drop Track function, byte 2 contains the address of the Track Reservation Table and byte 3 contains the half track number. For some functions, the function arguments are placed in the Message Buffer and only the function code appears in the Output Register.

MTR regularly scans the Output Register of each processor to determine if a request is present. When the request has been detected, analyzed, and processed, MTR clears the Output Register. The resident, after placing the request in the Output Register, waits for the Output Register to be cleared before proceeding.

Some functions require that information be returned by MTR to the requesting program: for example, the Request Track function (function number 6) returns a half track number to the requestor. MTR places any information to be sent to the requestor in the Message Buffer. The resident returns control to the requesting transient or overlay program when it detects that the Output Register has been cleared by MTR: the requesting program then reads the Message Buffer to obtain the required information.

The resident contains a routine called Process Request which handles the transmission of function requests to MTR. The Process Request routine uses locations 10 - 14 in peripheral processor memory as temporary storage for the request to be written in the Output Register. A peripheral processor program may utilize this routine by placing the arguments for the function in bytes 11 and 12, setting the A register with the function number, and executing a return jump to the Process Request routine at location 761. The Process Request routine will enter the function number in location 10 and write the contents of locations 10 - 14 in the Output Register. Control will be returned to the requesting program upon MTR's clearing the Output Register.

THE RESIDENT PROGRAM

When a pool processor program completes execution, it exits to location 100, which is the address of the resident idle loop. In this idle loop, the processor's Input Register is scanned at intervals of slightly greater than 125 microseconds until a request is found in the Input Register. The delay between successive scans avoids unnecessary memory and read pyramid conflicts. When a request is detected, the resident stores the routine name and the control point number. It then sends function code 17, Pause for Storage Relocation, to MTR and waits for MTR to clear the Output Register before continuing. Should MTR be in the process of relocating the storage assigned to this control point, the Output Register clear will be delayed until relocation is

complete. The resident then searches the RPL for the requested routine: if found, the package is read from the resident library into the processor's memory beginning at location 773, and resident turns control over to this routine by jumping to location 1000. If the routine name was not found in the RPL, resident then initiates a search of the PLD. If the routine is in the disk library, the resident loads it from the disk into its memory at location 773, and jumps to it to begin execution. If the routine is not found in the PLD, the resident enters the message "XXX NOT IN PPLIB" in the dayfile, and requests MTR to abort the job which called the routine. The resident then returns to its idle loop.

In loading a program from the disk, resident begins by reserving channel 0 via the appropriate MTR function request. Next, resident compares the track number of the requested routine with the current position of the disk as contained in the TRT pointer word for disk 0. Repositioning and/or head group switching is done only if necessary. Once the disk has been properly positioned, the sectors composing the desired routine are read into peripheral processor memory. The end of the routine is indicated when a short record (less than $100_8$ central memory words) is read. If a parity error is detected, the sector in which the error occurred is reread twice, each time at a different clipping level. Should these reads also fail, the resident enters the message "DISK 0 PARITY ERROR Gx Txxx Sxxx" in the dayfile and then stops (via a UJN 0 instruction). A dead start load is necessary to renew systems operation.

Several resident routines are used by transient and overlay programs. These routines are described below.

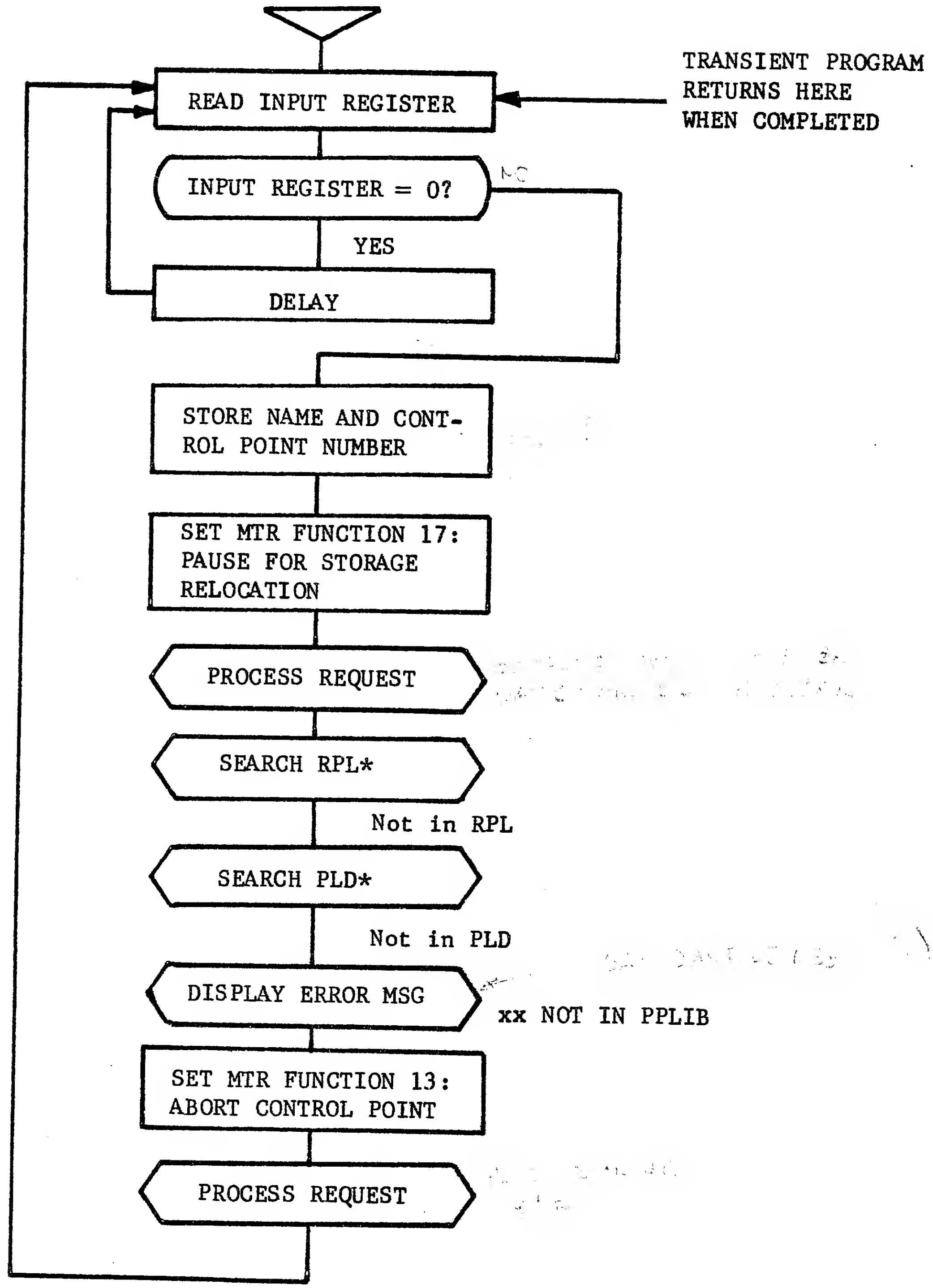
| Address | Routine | Entry Conditions | Description |
|---|---|---|---|
| 761 | Process Request | Function number in A register | Enters function number in location 10, and writes locations 10 - 14 to the Output Register. Exits when the Output Register has been cleared |

| Address | Routine | Entry Conditions | Description |
|---|---|---|---|
| 741 | Request Channel | Channel number in A register | Stores channel number in location 11, sets function code 2 in A register, and jumps to Process Request |
| 751 | Drop Channel | Channel number in A register | Stores channel number in location 11, sets function code 3 in A register, and jumps to Process Request |
| 531 | Dayfile Message | Message address in A register | Write message (less than 6 CM words, terminated by a zero byte) in Message Buffer, sets function code 1 in the A register, and jumps to Process Request |
| 701 | Position Disk* | Half track number in A register | Repositions heads and/or switches head groups as necessary (for disk 0 only) |
| 200 | Disk Parity Error Exit* | Half track number in location 6, sector number in location 7 | Enters error message in the dayfile and halts |
| 401 | Read Sector from Disk 0* | Read address in A register, half track number in location 6, sector number in location 7 | Reads one sector from disk 0 into memory at the designated address. Jumps to Disk Parity Error Exit if an error occurs. |

* Not a MTR function

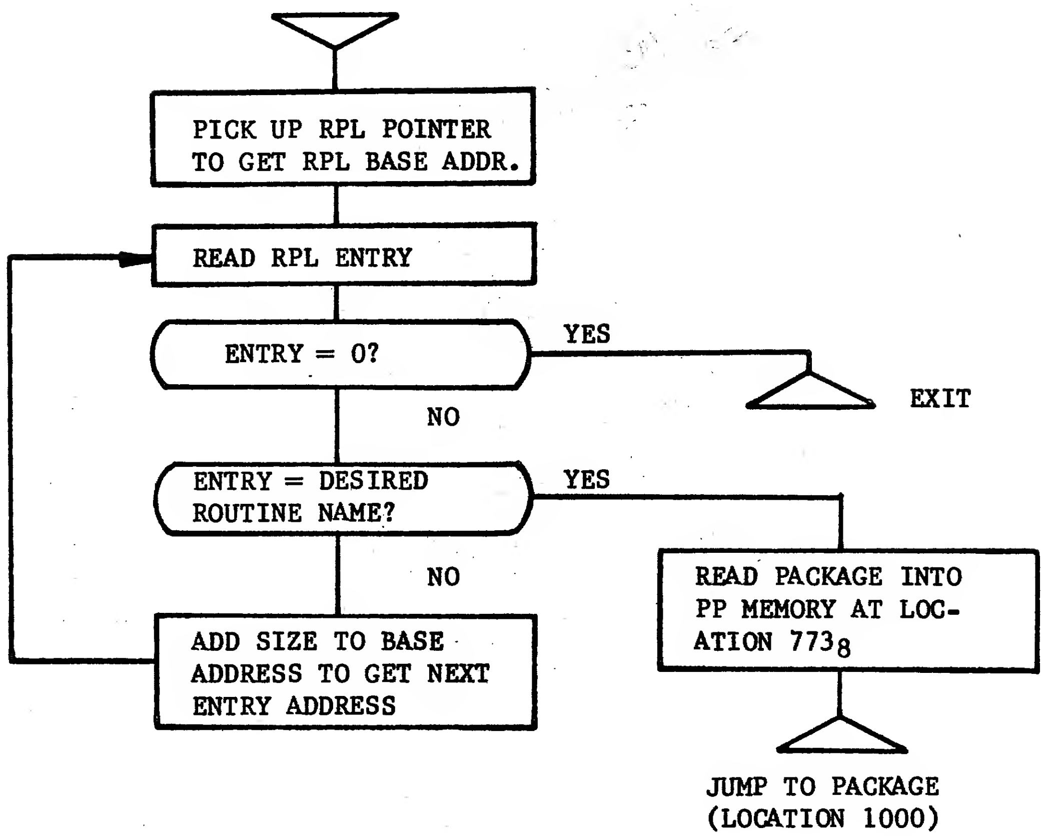
All of the foregoing routines are entered via a return jump instruction to the specified address except the Disk Parity Error Exit, which is entered via a long jump instruction.

PP RESIDENT: IDLE LOOP

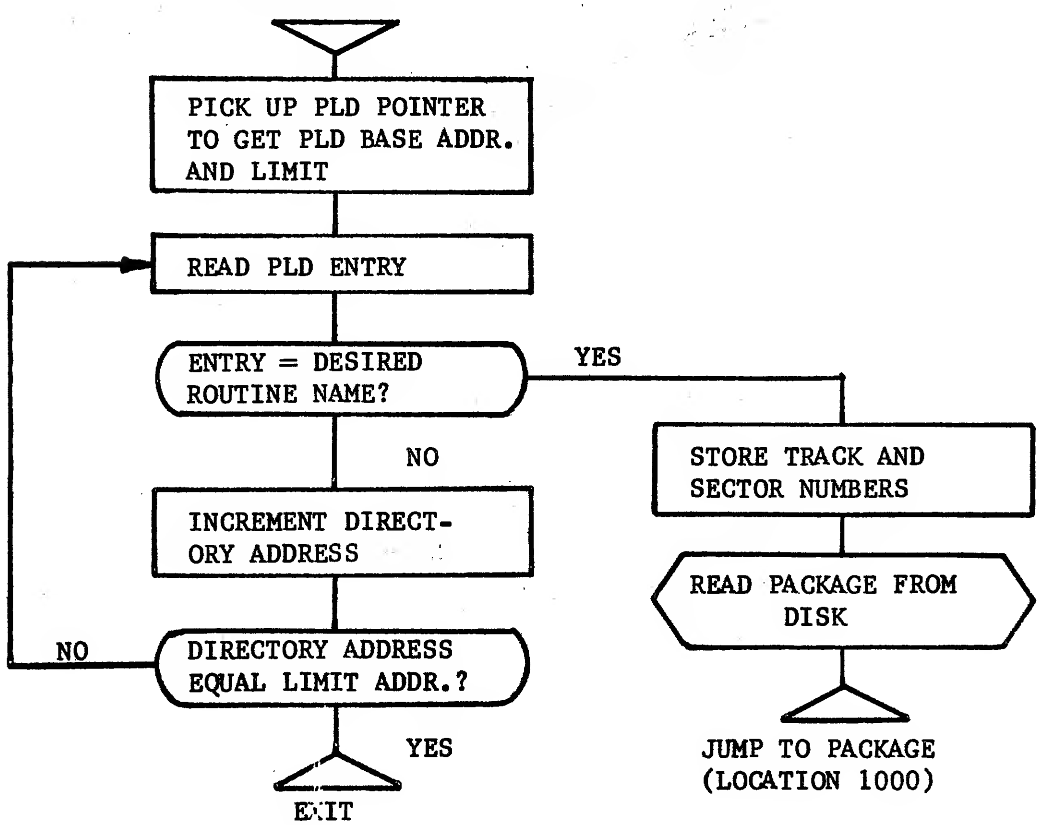



* IF FOUND, LOAD PROGRAM AND JUMP TO IT

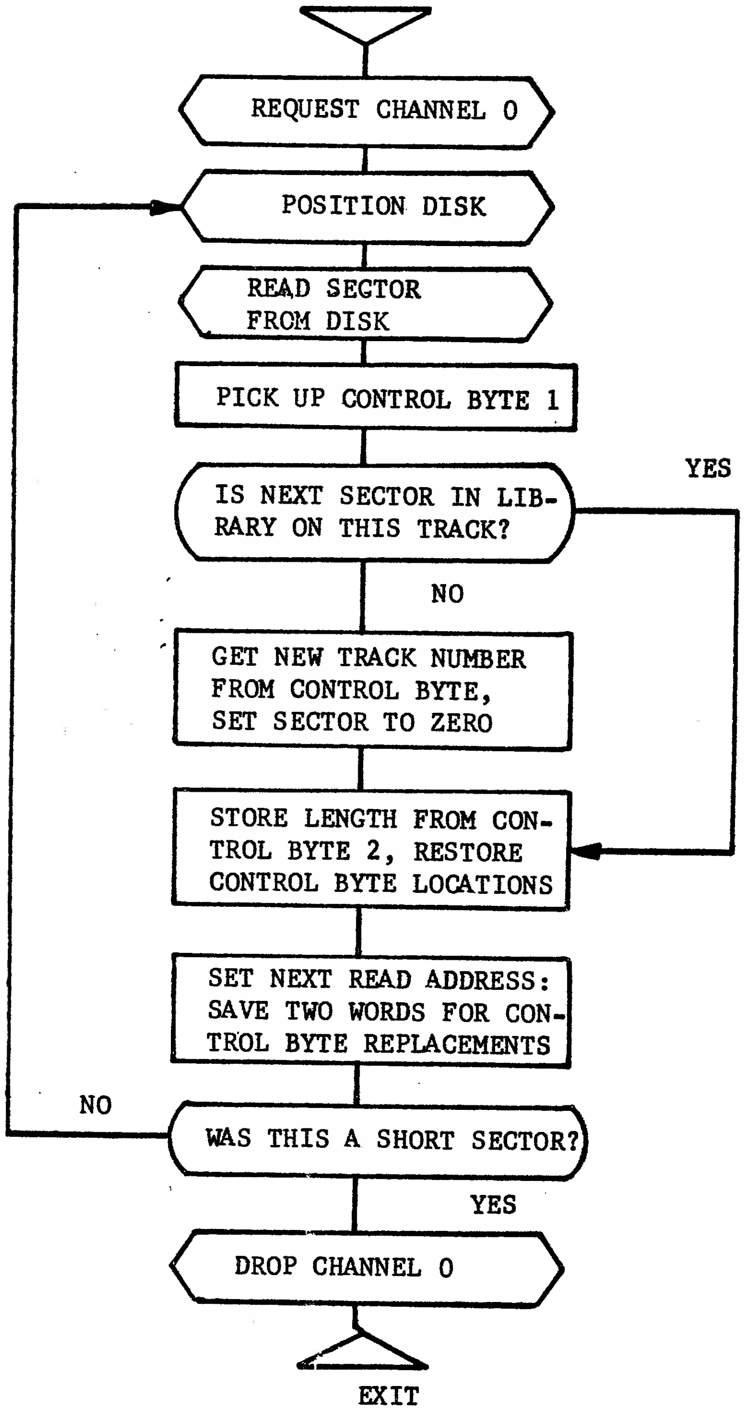
SEARCH RPL



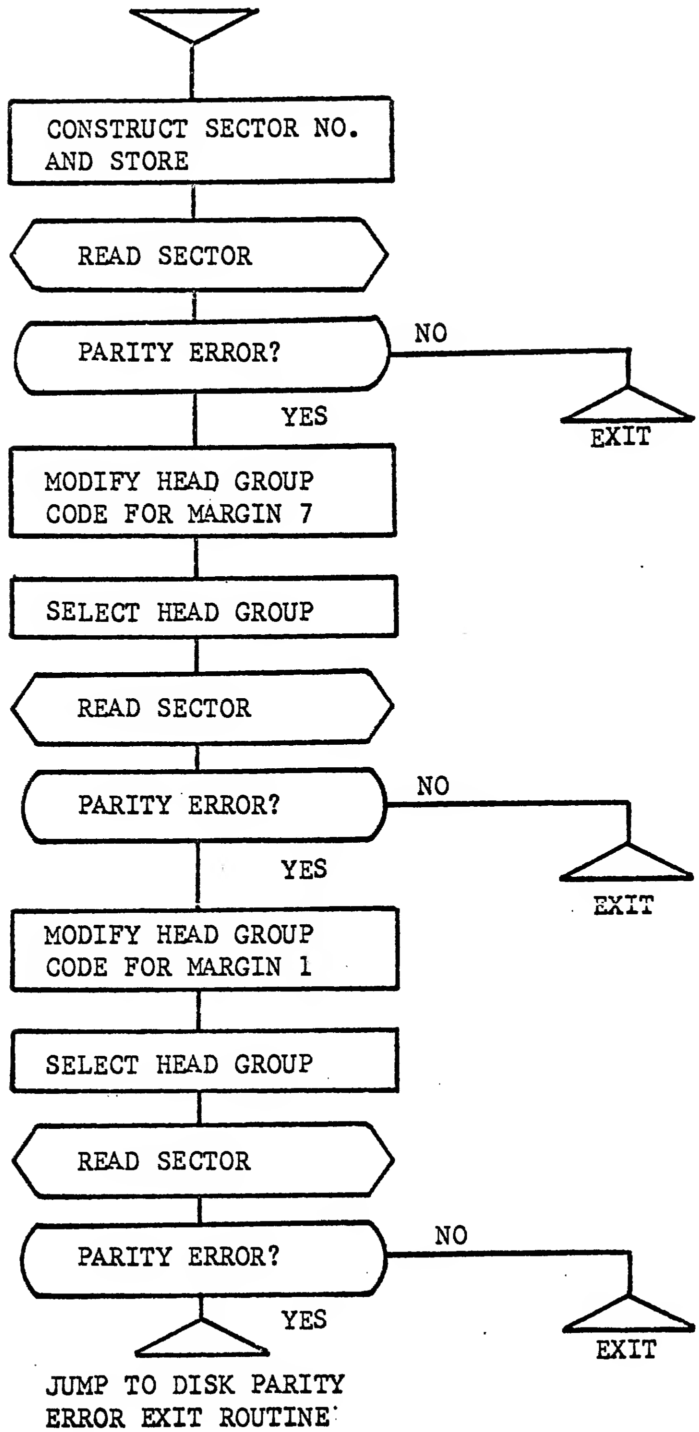

SEARCH PLD



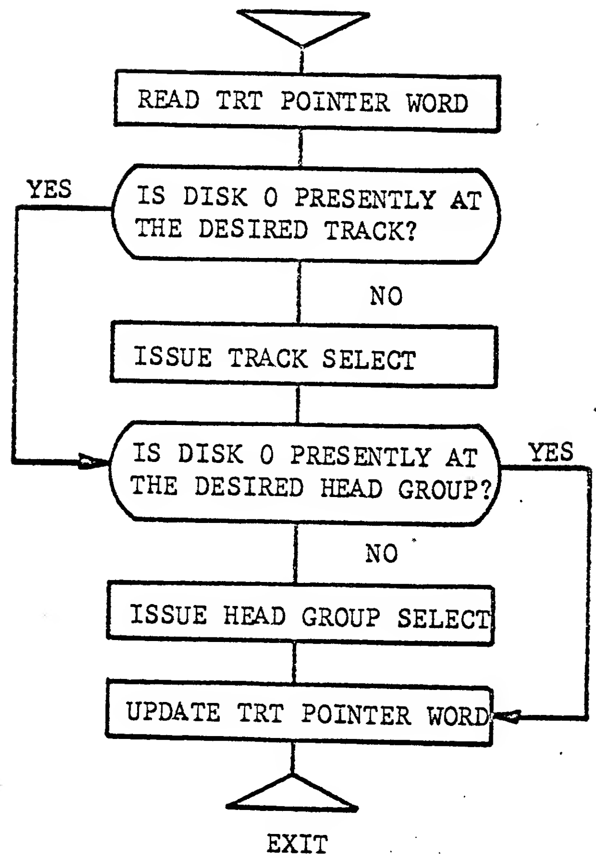
READ PACKAGE FROM DISK



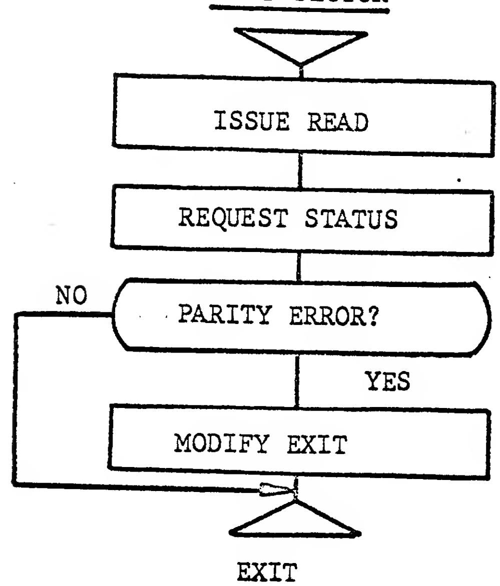
READ SECTOR FROM DISK



POSITION DISK

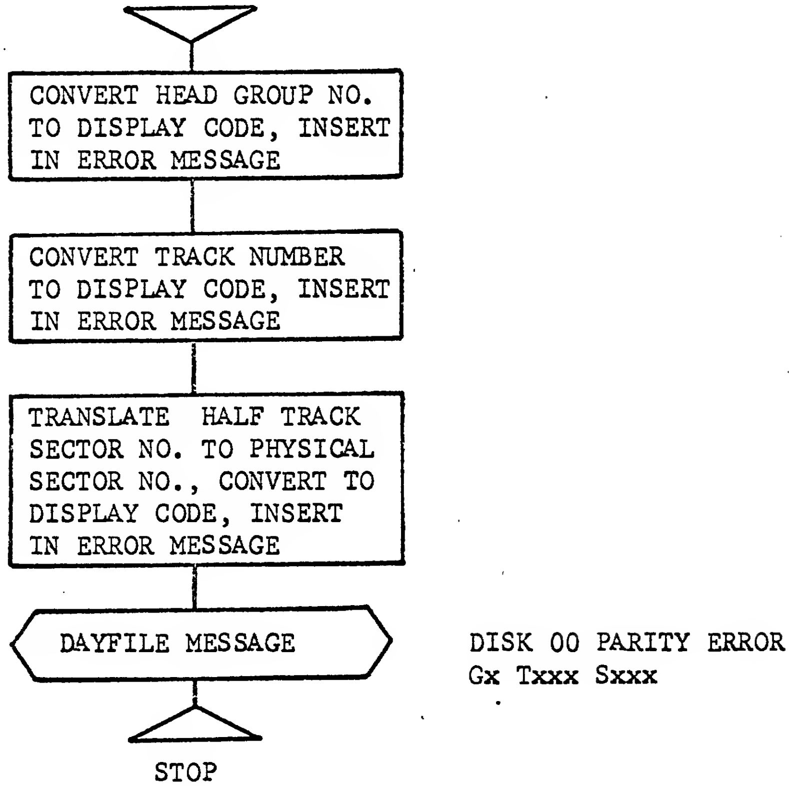



READ SECTOR

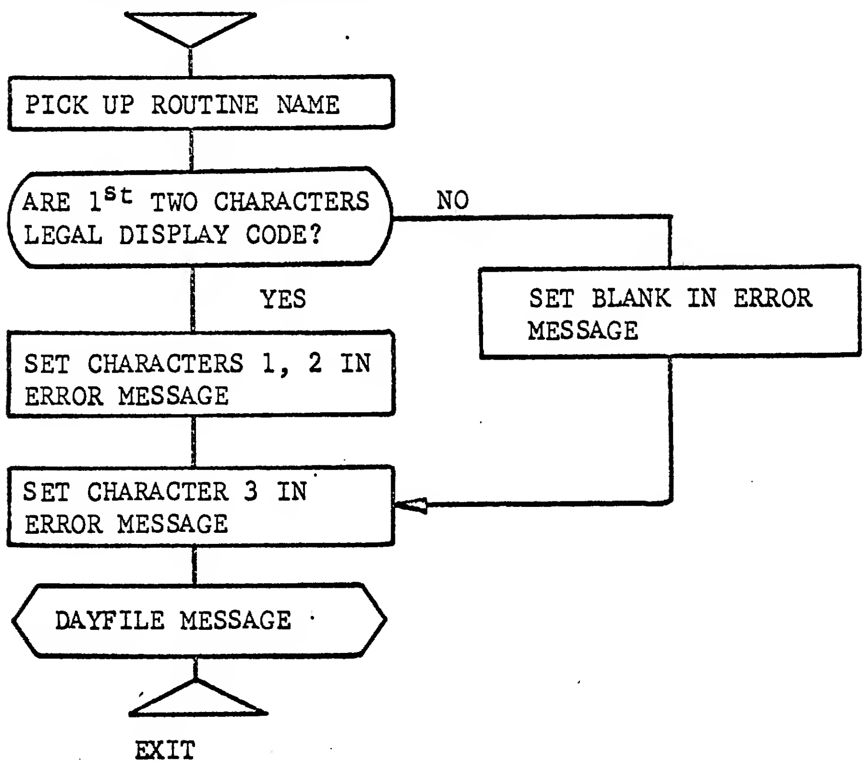


DISK PARITY ERROR EXIT

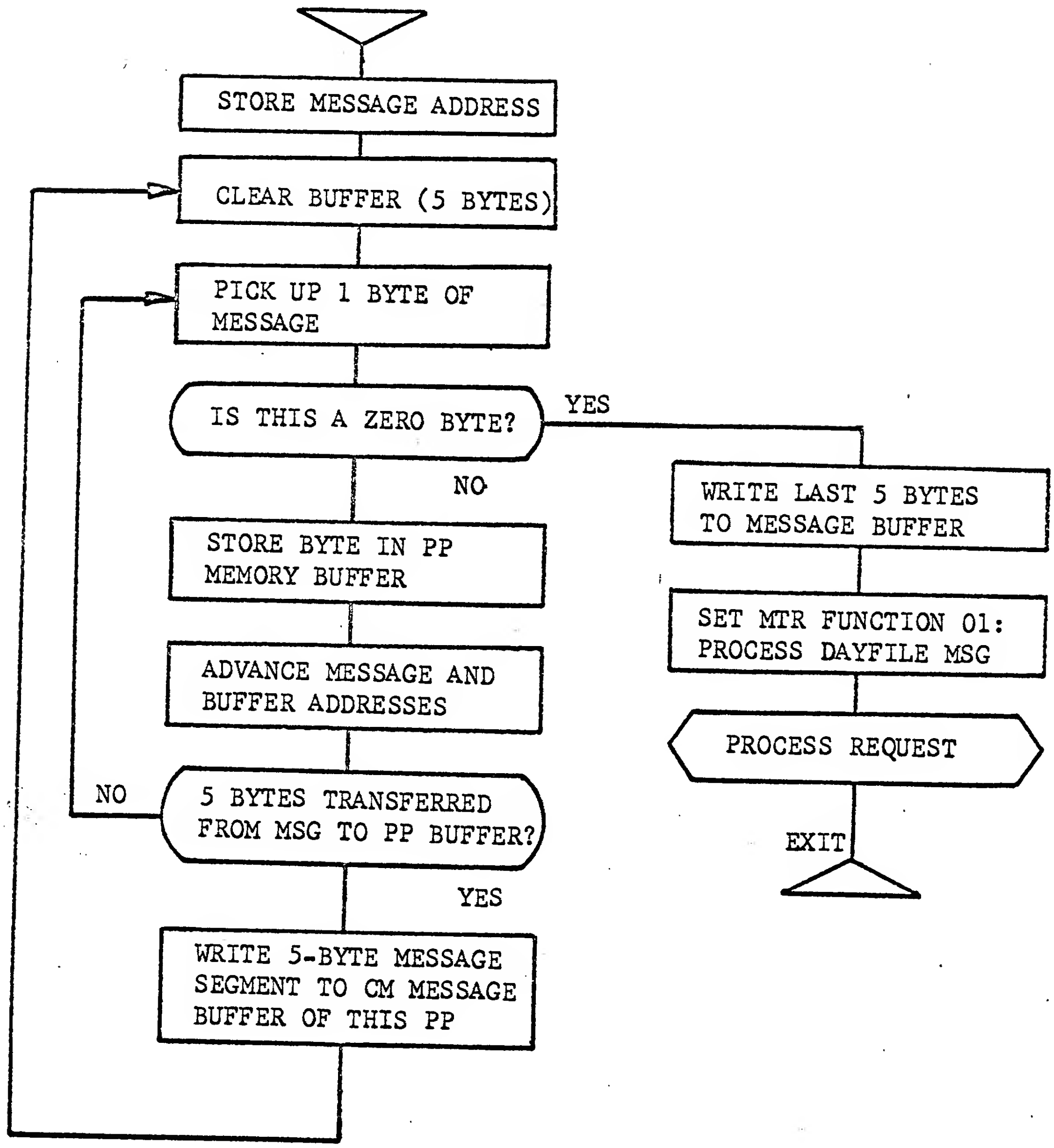
Used by transient programs and overlays as well as by PP Resident



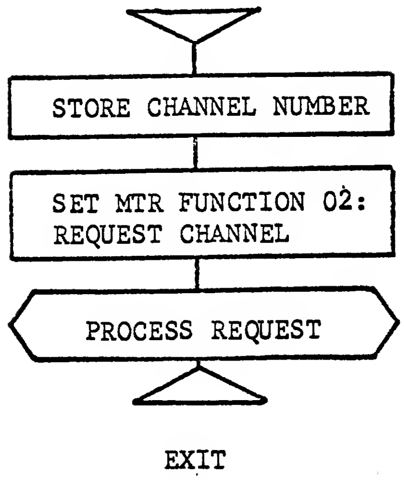

DISPLAY ERROR MESSAGE



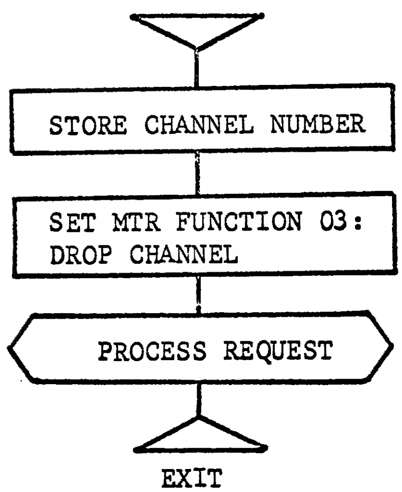
DAYFILE MESSAGE



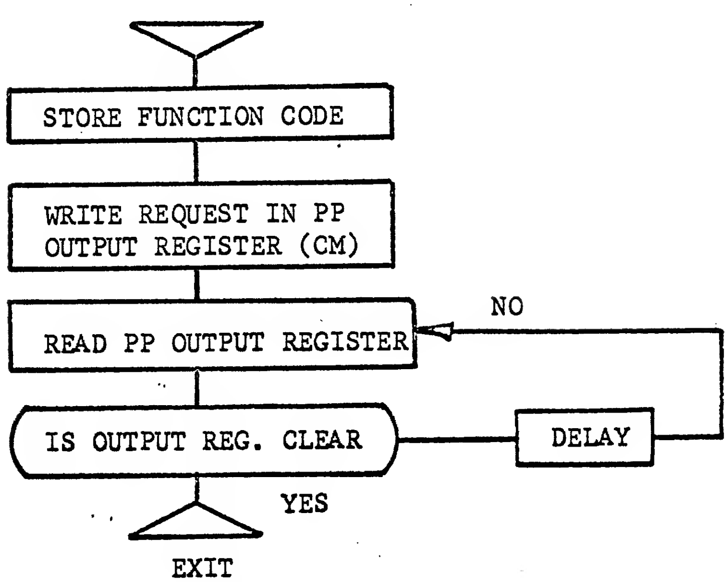
REQUEST CHANNEL




DROP CHANNEL




PROCESS REQUEST



CONTROL DATA CORPORATION

Development Division - Applications

# THE SYSTEM MONITOR, MTR

Chippewa Operating System

10/19/65
Rev. 1

# THE SYSTEM MONITOR, MTR

## CONTENTS

# THE SYSTEM MONITOR, MTR

## INTRODUCTION

The monitor, or executive, of the Chippewa Operating System is the MTR program, which permanently resides in peripheral processor 0. Among the functions performed by MTR is the allocation of the physical components of the system to various users. The components controlled by MTR include:

- pool processors
- peripheral equipment - tapes, printers, card readers, etc.
- data channels
- disk tracks
- central memory

MTR directs the loading and initiates the execution of central processor programs, monitors central processor programs for I/O requests and assigns these requests to available peripheral processors, and monitors peripheral processor programs for function requests. MTR maintains the time accounting in the system and is responsible for the maintenance of the dayfile.

## THE CONTROL POINT CONCEPT

In a multiprogrammed multiprocessor such as the 6600 system, central memory is shared by a number of users. In addition to the active and inactive central processor programs residing in central memory, many peripheral processor programs require central memory buffers. The allocation of central storage to these various users is a function which the operating system can handle in one of two ways:

1. Storage can be allocated to a number of users limited only by

the amount of memory available. This assures the maximum utilization of central memory, but requires an elaborate bookkeeping system. In particular, the manipulation of the variable length tables required, and the relocation of storage to avoid arriving at a "patchquilt" of unallocated memory locations as jobs complete, present interesting design problems.

2. Storage can be allocated to a fixed number of users. If the limit is properly selected, losses in memory utilization efficiency will be minimal. In this method, control of storage allocation and relocation is greatly simplified.

For many job mixes, system throughput is not materially affected by the use of one or the other of the above methods.

The Chippewa Operating System uses the second method described. In the Chippewa Operating System, central memory may be simultaneously shared by up to seven users. For each of the seven users sharing central memory, there is an area in the central memory resident called the control point area. As each user is assigned storage, pertinent information about the user is entered in the control point area: as execution proceeds, entries are made in the control point area to reflect the current status of the user.

The seven control point areas are each $200_8$ central memory locations in length, and occupy a portion of the central memory resident between locations 0200 and 1777. The control point areas are numbered one through seven in accordance with their relative (to one another) locations in central memory resident: control point 1 refers to the control point area in locations 0200 - 0377, control point 2 refers to the control point area in locations 0400 - 0577, and so forth. If the information about a user is contained in a given control point area, the user is said to be assigned to that control point.

The user assigned to a control point may be a peripheral processor program, a central processor program, or both: the last case occurs when a central processor program employs a peripheral processor program to perform an input-output operation. Control point assignments are required not only for external users (i.e., jobs) but for many of the operating system programs as well. Thus, the system program which transfers jobs from the card reader to the disk (1LJ) must be assigned to a control point, since a central memory buffer is required.

In many instances, the system packages READ, NEXT, and PRINT will each be assigned to a control point. (READ = load a job from the card reader and place it on the disk, NEXT = load a job from the disk, and PRINT = print the output of a job.) Each of these packages requires central memory space: the total space required by all three packages is $10300_8$ locations. These three packages plus the central memory resident occupy $24300_8$ or about $10400_{10}$ locations. This leaves approximately $120,000_{10}$ locations to be shared by users assigned to the remaining four control points, or about $30,000_{10}$ locations per control point - certainly a reasonable amount of storage for many jobs.

The control point area is illustrated in figure 1. The first sixteen words of the control point area contain the exchange jump package. If the user assigned to the control point is a peripheral processor program, no use is made of this exchange jump package insofar as this user is concerned. If the user assigned to this control point is a central processor program, this package is set with the appropriate values of P, RA, FL and EM when the program is initiated: as central processor programs are interrupted and restarted, the exchange jump packages for other central processor programs appear here.

Regardless of whether the user assigned to this control point is a central processor program or a peripheral processor program, the storage allocated

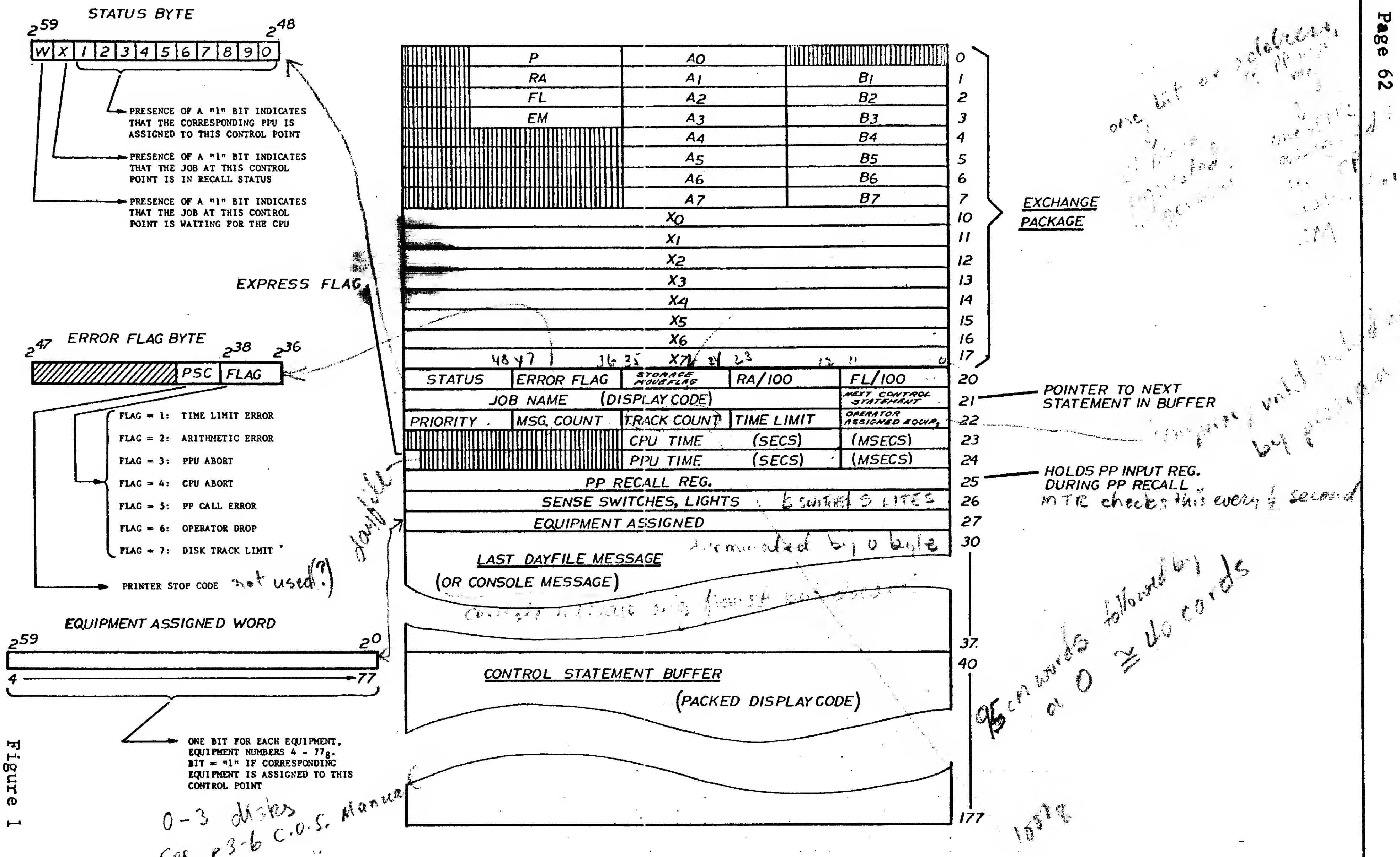



Figure 1

# CONTROL POINT AREAS

CHIPPEWA OPERATING SYSTEM

| CONTROL DATA CORPORATION SOFTWARE DOCUMENT | | | |
|---|---|---|---|
| SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☒ | DOCUMENT CLASS: TRAINING | MACH TYPE: 6000 | PROJECT NO |
| | DOCUMENT TITLE: CONTROL POINT AREA | | PROJECT MGR. |
| | PAGE 1 OF 1 | | PROJECT NAME |
| | NUMBER | ISSUE DATE: 11-1-65 | TASK NO. |
| | DRAWN BY: MH | DATE: 11/65 | TASK NAME |

DOCUMENT ABSTRACT

| REV | APPROVED | DATE | REV | APPROVED | DATE |
|---|---|---|---|---|---|
| | | | | | |

is always defined by the values of RA and FL in bytes 4 and 5, respectively, of location $20_8$ within the control point area. Note that these values are in hundreds (upper 12 bits of an 18-bit address).

The control point number is often maintained in the low-order three bits of a byte. On many occasions, the system derives the control point area address by shifting the control point number left 7 places from its low-order bit positions. For example, a routine might pick up a byte containing the number of control point 2, which would appear as 0002: shifting this left 7 places, we obtain 0400, the beginning address of the control point 2 area.

## MTR: DEAD START HOUSEKEEPING, THE CP IDLE PROGRAM, AND CONTROL POINT 0

During the loading of the system tape, the lower portion of the central memory resident is initialized by reading a series of records totalling $5000_8$ CM words into central memory beginning at location 0. This initialization process sets the first entry in the FNT/FST with the file name DAYFILE and the file type COMMON.

When the loader releases peripheral processor zero to MTR, MTR obtains the next available track number from the Track Reservation Table for disk 0. This half track number is set in the Beginning Track (byte 2) and Current Track (byte 3) bytes of the FST entry for the dayfile. Byte 1, the Equipment Number, is set to zero as is byte 4, the Current Sector byte. The Buffer Status byte (byte 5) is set to 1, indicating that this file is not reserved.

Once the FNT/FST entry for the dayfile has been completed, MTR issues an exchange jump to the central processor idle program. This idle program executes a jump to relative location 2, which contains a stop instruction, and thus halts the central processor with $P \neq 0$. The function of the idle program is to keep $P \neq 0$ in all cases except in the case of an error exit from a central processor program.

The idle program is a central processor program, and as such must be

assigned to a control point. A pseudo control point, called control point zero, is used for this purpose. Referring to the control point area illustration (figure 1), note that relative locations $21_8$ and $20_8$ contain, respectively, the job name and the job status. The control point area for control point zero is assumed to start at location 0 in central memory: central memory locations $21_8$ and $20_8$ (absolute) contain the job name and the job status for control point zero. These are the only locations in this portion of the resident which are actually a part of the control point zero area: the exchange jump package for the idle program begins at location 2040. Location $21_8$ contains MONITOR as the job name. Byte 1 of location $20_8$ contains the job status: the low order bits of this byte are used to indicate the assignment of peripheral processors to a control point. For control point zero, the status byte contains 0003, indicating that processor 0 (MTR) and processor 9 (DSD) are assigned to this control point.

The use of the pseudo control point zero is a mechanism simplifying the manner in which MTR controls the assignment of jobs to the central processor. The reason for using location 0 as the start of the control point area for control point zero is evident when we remember that the address of a control point may be obtained from its control point number by shifting the control point number left seven places.

After initiating the central processor idle program, MTR enters its master loop.

## MTR: USE OF LOW CORE LOCATIONS

MTR uses low core locations 26 - 77 to maintain various flags, pointers, and special-purpose buffers: these are illustrated in figure 2. Locations 75 - 77 contain the Input Register, Output Register, and Message Buffer pointers for the peripheral processor zero communication area. A five-byte area

## MTR: USE OF LOW CORE

| Address | Contents | Use |
|---|---|---|
| 77 | MESSAGE BUFFER POINTER | POINTERS FOR PPO COMMUNICATION AREA |
| 76 | OUTPUT REGISTER POINTER | |
| 75 | INPUT REGISTER POINTER | |
| 74 – 70 | DATE LINE BUFFER | USED TO BUFFER TIME-DATE LINES FROM CM LOCATIONS 30 - 37 |
| 67 | SECOND COUNT | |
| 66 | MILLISECOND COUNT | |
| 65 | CLOCK PHASE | |
| 64 | DAYFILE DUMP FLAG | |
| 63 | LAST SECOND COUNT | |
| 62 | NEXT PP INPUT ADDRESS | INPUT REGISTER ADDRESS OF A FREE POOL PROCESSOR - ZERO IF NONE AVAILABLE |
| 61 | COMPLETE DAYFILE FLAG | |
| 60 | ACTIVE CONTROL POINT ADDR. | POINTS TO CONTROL POINT AREA TO WHICH THE CENTRAL PROCESSOR IS ASSIGNED |
| 57 – 52 | ACTIVE CONTROL POINT STACK | ONE PP WORD FOR EACH CONTROL POINT, 1 - 7 |
| 51 | MOVE STORAGE FLAG | |
| 50 | TEMPORARY | OFTEN USED TO TRANSMIT A PP OUTPUT REGISTER ADDRESS BETWEEN MTR ROUTINES |
| 47 | TEMPORARY | |
| 46 – 30 | CHANNEL STATUS BUFFER | 15 PP WORDS, USED IN UPDATING CHANNEL STATUS TABLE |
| 27 | CP ADVANCE COUNTER | |
| 26 | NEXT CP NUMBER | USED IN ADVANCING CONTROL POINT SCAN |

Figure 2

consisting of locations 70 through 74 is used to buffer the time line from the time-date area in central memory locations 30 - 37. This central memory area contains the time and, optionally, the date: it is initialized via the DSD keyboard entry "TIME". The first word of this central resident area contains the time: this word is read into locations 70 - 74 whenever the time is to be advanced or entered in a dayfile message. Locations 63, 65, 66, and 67 contain counts used in advancing the clock and in computing time charges to a control point.

Location 64 contains the Dayfile Dump Flag, which, when set, indicates that the full sectors in the dayfile buffer are being dumped to the disk and also indicates which phase of the dumping process is to be executed next. Location 61 contains another dayfile related flag, the Complete Dayfile Flag, which is used in insuring that dayfile messages for a specific job are dumped to the disk at the end of a job.

Location 62 contains the address of the Input Register of a free pool processor: this processor will be assigned by MTR to the next peripheral processor task. If all pool processors are busy, this location contains zero.

Locations 60 and 52 - 57 hold the control point stack. Location 60 represents the top of the stack and contains the address of the control point area for the program currently being executed by the central processor: if this location contains zero, the central processor is unassigned (i.e., is assigned to pseudo control point zero, the control point for the idle program). Control points representing programs waiting for the central processor are stacked in locations 52 - 57.

Location 51 contains a Move Storage Flag, used when storage is being reallocated to control points. Location 50 is a temporary storage area: it is often used to transmit the Output Register address of a peripheral processor between MTR routines.

Locations 30 - 46 provide a buffer used by MTR in updating the Channel Status Table. Locations 26 and 27 are used by MTR in advancing the control point scan: location 27 contains a count used in determining the time interval between successive scans, and location 26 contains the number of the control point to be processed on the next scan.

The remaining low core locations, 01 - 25, are used for a variety of temporary storage needs. For example, locations 10 - 14 are used at various times to hold a peripheral processor's Output Register, the status word from a control point area, a TRT pointer, and a variety of other quantities.

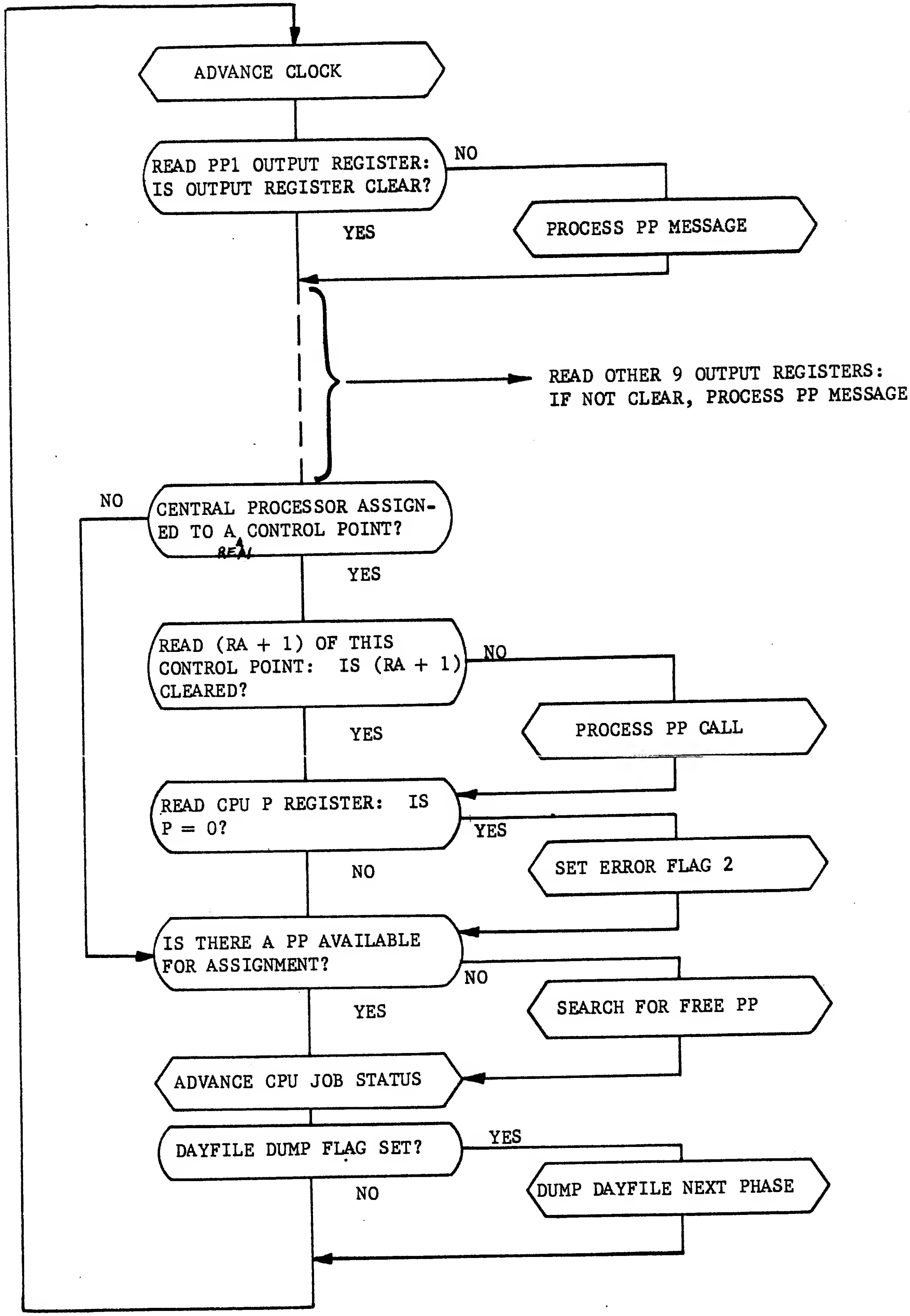
## MTR: MASTER LOOP

The MTR Master Loop is illustrated in figure 3. This loop, from which all MTR routines are entered (either directly or indirectly), performs the following four major functions:

- Advances the system clock
- Monitors peripheral processors for function requests
- Monitors the central processor program currently being executed for I/O requests and normal or abnormal exit conditions
- Examines one of the seven control points for PP or CP recall status and may initiate another central processor program: if the control point is inactive, the 1AJ routine is called to bring a job from the disk to this control point.

The time between successive scans is primarily a function of the number and type of requests serviced during a scan. In any case, the fourth function mentioned above (Advance CPU Job Status) is performed at intervals of no less than 64 milliseconds.

The Advance Clock routine updates the system clock, which is stored in location 30 in central memory resident. This word generally has the format




*MTR MASTER LOOP*

Figure 3

"sp HR . MN . SC .", where HR, MN, and SC are each two display code digits representing, respectively, hours, minutes, and seconds.

On each pass through the loop, MTR reads the Output Register of each peripheral processor, including its own. All requests to MTR from peripheral processor programs are transmitted in the form of function codes placed in the requesting processor's Output Register. When MTR finds a request in an Output Register (i.e., Output Register not cleared), it performs a table look-up for the routine corresponding to the function number, and jumps to that routine. If the request can be executed, the routine clears the Output Register before exiting back to the master loop: if the request cannot be executed, the routine exits to the master loop without clearing the Output Register. In the latter case, MTR will pick up the request again on its next trip through the master loop, and attempt to execute the request once again.

The functions performed by MTR for peripheral processor programs are listed below. The flow chart page numbers refer to the attached flow charts: memory addresses refer to the version of MTR dated 10/15/64.

Registers p 44

| Function Number | Starting Address | Flow Chart Page No. | Function |
|---|---|---|---|
| 1 | 1500 | A-3 | Process Dayfile Message |
| 2 | 2000 | A-4 | Request Channel |
| 3 | 2040 | A-4 | Drop Channel |
| 4 | 2440 | A-4 | Assign PP Time |
| 5 | 1560 | A-5 | Monitor Step Control |
| 6 | 2200 | A-5 | Request Disk Track |
| 7 | 2300 | A-5 | Drop Disk Track |
| 10 | 4300 | A-6 | Request Storage |
| 11 | 1300 | A-7 | Complete Dayfile |

| Function Number | Starting Address | Flow Chart Page No. | Function |
|---|---|---|---|
| 12 | 3730 | A-7 | Release PP |
| 13 | 4040 | A-7 | Abort Control Point |
| 14 | 3600 | A-8 | Enter New Time Limit |
| 15 | 2600 | A-8 | Request Central Processor |
| 16 | 3760 | A-8 | Release Central Processor |
| 17 | 5200 | A-9 | Pause for Storage Relocation |
| 20 | 4640 | A-9 | Request Peripheral Processor |
| 21 | 2750 | A-9 | Recall Central Processor |
| 22 | 5600 | A-10 | Request Equipment |
| 23 | 5240 | A-10 | Drop Equipment |
| 24 | 3240 | A-10 | Request Priority |
| 25 | 3630 | A-11 | Request Exit Mode |
| 26 | 3030 | - | Reserved for Future Use |
| 27 | 3100 | A-11 | Toggle Simulator |
| 30 | 2160 | A-12 | Operator Drop |
| 31 | 4200 | A-12 | Ready Tape |
| 32 | 4240 | A-12 | Drop Tape |
| 33 | 6100 | A-12 | Assign Equipment |
| 34 - 37 | 3030 | - | Reserved for Future Use |

After servicing any peripheral processor requests which may have been present, MTR proceeds to determine if any action is required by the central processor. To determine if the central processor is executing a program, MTR looks at the top of the control point stack (location 60 in processor 0's memory). If this location contains zero, the central processor is idle: if the contents of this location are non-zero, then the central processor is currently executing a program. The entries in the stack are control point

addresses: thus, location 60 contains the address of the control point area for the program currently being executed by the central processor. MTR adds $20_8$ to this address to form the address of the Status word in the control point area (see figure 1), reads the Status word and extracts byte 4, which contains the reference address in hundreds. MTR then reads the contents of RA + 1 to determine if the central processor program has issued a request.

If the contents of RA + 1 are not zero, MTR jumps to a routine to process the request. If RA + 1 contains END or RCL, another central processor program is initiated in place of the current one: if RA + 1 contains ABT or if the request in RA + 1 does not begin with a letter, the appropriate error flag bit is set in byte 2, location $20_8$, of the control point area. If RA + 1 contains a legitimate PP call, MTR places the call and the control point number of the requestor in the Input Register of an available pool processor and assigns the processor to this control point by setting the appropriate bit in byte one of the Status word. After processing the call, (RA + 1) is cleared to inform the central processor that the request has been processed, and control is then returned to the master loop. If the call was END, ABT, or illegal, or if the request could not be processed at this time (no free pool processor), the routine exits to the master loop without clearing RA + 1. The subroutine which processes central processor requests for peripheral programs is entitled "Process PP Call". Its starting address is 2700, and it appears on page A-14 of the attached flow charts.

After processing the central processor program request, MTR reads the central processor P register. If P contains zero, it is assumed that an error exit has occurred, and MTR sets the appropriate error flag in byte 2 of the Status word in location $20_8$ of the control point area.

MTR then looks at location 62 in its memory to see if it has a pool processor available for assignment. If this location contains zero (no

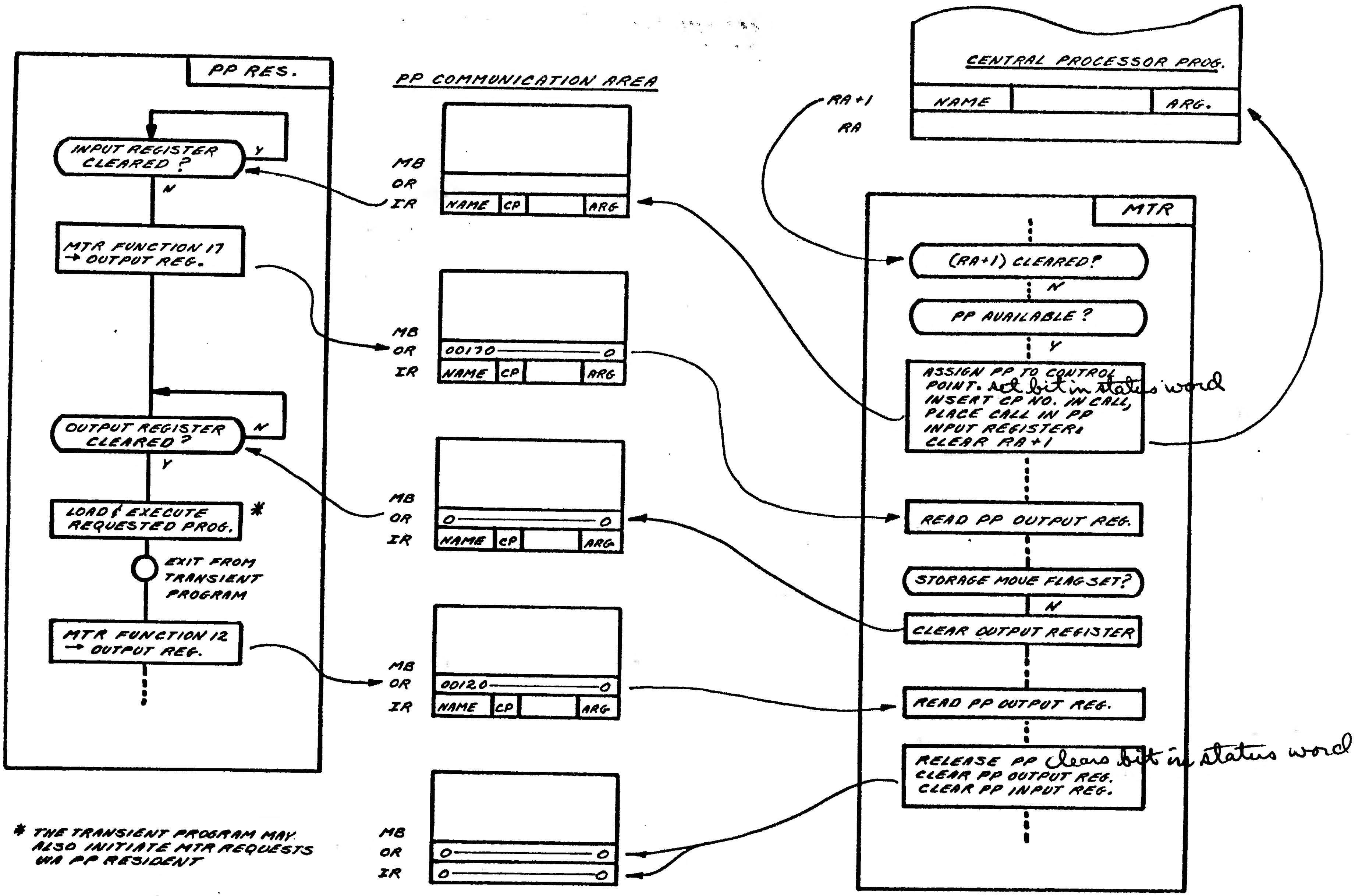
processor available), MTR scans the Input Registers of processors 1 - 8 and writes the address of the first cleared Input Register in location 62.

MTR next examines one of the seven control points and determines if the control point is in recall status. If it is, then this program may be re-initiated by MTR, depending upon its priority. If the control point is in-active, MTR directs the loading of another job at this control point. MTR scans only one control point on each pass through the master loop: the number of the control point most recently scanned is maintained in location 26 of MTR's memory. The MTR subroutine which performs this processing is entitled "Advance CPU Job Status" and is shown on page A-13 of the attached flow charts. This subroutine will be discussed at greater length during the description of the control point stack.

If dayfile dumping is not in process, MTR returns to the beginning of the loop and begins its scan once more.

As a review of CPU - MTR - PP communication, the sequence which takes place when a central processor program requests that a task be performed by a peripheral processor program is described below. (Refer to figure 4.)

1. The central processor program requests a peripheral processor by writing the routine name (three display code characters), left-justified, in location 1 of its program. The address of any parameters required are written in the low-order bits of this location.
2. MTR examines the contents of RA + 1 during its master loop: if (RA + 1) is non-zero, MTR jumps to a subroutine to process the call. This subroutine inserts the control point number of the requesting job in the low-order three bits of byte 2 of the word read from RA + 1 and then writes this word in the Input Register of a free pool processor. The routine also sets the bit corresponding to the processor in byte 1 of word $20_8$ in the control point area.




* THE TRANSIENT PROGRAM MAY ALSO INITIATE MTR REQUESTS VIA PP RESIDENT

Figure 4

| CONTROL DATA CORPORATION SOFTWARE DOCUMENT | | | |
|---|---|---|---|
| SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☒ | DOCUMENT CLASS TRAINING; MACH TYPE 6000 | PROJECT NO. | DOCUMENT ABSTRACT |
| | DOCUMENT TITLE CP-MTR-PP COMM. | PROJECT MGR | CP PROGRAM - MTR - PP RES. COMMUNICATION |
| | PAGE OF | PROJECT NAME | CHIPPEWA OPERATING SYSTEM |
| | NUMBER; ISSUE DATE | TASK NO. | |
| | DRAWN BY HM; DATE 9-15 | TASK NAME | |

| REV | APPROVED | DATE | REV | APPROVED | DATE |
|---|---|---|---|---|---|

CA 127-5

3. When the peripheral processor resident finds the routine name in its Input Register, it asks MTR if the storage assigned to this control point is to be relocated by issuing the appropriate function request (function code 17).
4. If the storage assigned to this control point is to be relocated, MTR will delay the execution of the requested transient program by not clearing the Output Register of the peripheral processor until relocation is completed. If no storage relocation is to be done, MTR clears the Output Register immediately upon recognizing the function request.
5. When the Output Register has been cleared by MTR, the resident proceeds to load the requested program either from the resident library or the disk library, and then transfers control to it.
6. The transient program and any overlays it may use may also communicate with MTR by using the resident subroutines to transmit function requests to MTR. These programs may also communicate directly with the central program by adding the parameter address (held in the Input Register) to the value of RA from the control point area in order to obtain the absolute address of information within the central processor program.
7. When the transient programs completes execution, it sends (via peripheral resident) a Release PP function request (function code 12) to MTR. MTR clears the processor's Input Register and Output Register, and clears the bit in byte 1 of the Status word (control point area, location $20_8$) corresponding to this processor.

MTR: JOB INITIATION

Once the system loader has released control of the peripheral processors to their respective programs, the pool processors begin spinning in their idle loops while MTR and DSD, after performing some initial housekeeping, enter their master loops. To initiate job loading and execution in the system, the operator may use the DSD keyboard entry "AUTO.". This assigns routines to control points as follows:

| Control Point | Routine |
|---|---|
| 1 | 1LJ (READ) |
| 2 | 1DJ (PRINT) |
| 3 | 1BJ (NEXT) |
| 4 | 1BJ (NEXT) |
| 5 | 1BJ (NEXT) |
| 6 | 1BJ (NEXT) |

DSD accomplishes this assignment by placing the routine name and control point number in the first two bytes of its Message Buffer and (via peripheral resident) issuing a Request PP function (function code 20) to MTR. MTR assigns a processor to the control point by writing the routine name and control point number in the Input Register of a free pool processor, and then setting the appropriate bit in byte one of the control point area status word.

The READ package (1LJ and its overlays) brings jobs in from the card reader and places them on the disk. It enters the job name as the file name in the FNT/FST table, inserts the priority from the job card in the FNT entry, and sets the file type to INPUT.

The NEXT package (1BJ and its overlays) loads a job from the disk to the control point to which it (NEXT) is assigned. 1BJ searches the FNT for the highest priority unassigned file of type INPUT. The file name (job name from the job card) is entered as the job name in the control point area. The file name is changed to INPUT, the file type changed to LOCAL, and the file assigned to this control point. The priority is placed in the control point area by

1BJ via an MTR function request (function code 24). 1BJ then calls overlays to read the first record from the file into the control statement buffer in the control point area. This record, which may be up to a full sector in length, contains the control cards for this job. The Next Control Statement pointer in the control point area is then initialized.

An overlay is called to complete job card translation. The time and the field length from the job card are inserted in the control point area by 1BJ via MTR function requests 10 and 14, respectively. (When 1BJ was initiated, it requested storage from MTR, who set the values of RA and FL in the control point area. Central storage is used by 1BJ for a buffer area: when the job is brought from the disk to the control point, the storage required by the job is requested by 1BJ. This storage is considered by MTR to be a replacement for, and not an addition to, the storage originally assigned to the control point.) In processing the storage request, MTR may have to relocate storage assigned to other (higher) control points. After this relocation is performed, MTR sets the value of FL in the control point area, both in byte 4 of location 20 and in the exchange jump package.

When 1BJ has completed its function, it requests MTR to release the peripheral processor. MTR then clears the processor's Input and Output Registers, and clears the appropriate bit in byte 1 of the status word in the control point area. This byte then contains zero. The processing performed by 1BJ has resulted in the control point area being set as follows:

- the job name, time limit, priority, and field length have been set.
- the pointer to the next control statement has been set to address whatever control card followed the job card.
- byte 1 of the Status word (location $20_8$ of the control point area is zero.

(Note; the control point areas are cleared during loading of the system and are also cleared each time a job is dropped from a control point.)

Once 1BJ has brought a job to a control point, further action concerning that job is initiated by MTR. This action will be described shortly: first we shall discuss the status of a job relative to the central processor.

MTR: JOB STATUS AND THE CONTROL POINT STACK

The status of a job is defined by the setting of bits $2^{11}$ and $2^{10}$ in byte one of the control point status word, and by the presence or absence in the control point stack of the control point address for the job. The $2^{10}$ bit is the X, or recall, flag. This flag is set when MTR detects RCL in RA + 1 of the program being executed by the central processor. The $2^{11}$ bit is the W, or wait, flag. This flag is set by various MTR routines to indicate that the job at the associated control point is waiting for the central processor. We may have two queues of jobs waiting for the central processor: one queue consists of jobs in the control point stack, and the other consists of jobs in W status. The top of the control point stack (location 60 in PP0 memory) represents the job currently being executed by the central processor. The remaining entries in the stack represent jobs interrupted because of the entry of a higher priority job into the system.

Whenever MTR sets the W flag for a job at a control point, a subroutine called Search for CP Priority is called. The flow chart for this subroutine appears on page A-16 of the attached flow charts. This subroutine checks the status of control points beginning with control point one. If the W flag at a control point is set, MTR compares the priority of this job with the priority of the job currently being executed by the central processor. If the job at the control point with the W flag set (i.e., the first control point found in wait status) has a higher priority than the job currently being executed, the routine pushes down the stack and inserts the control point address of the new job at the top of the stack, clears the W flag in the control point area, and

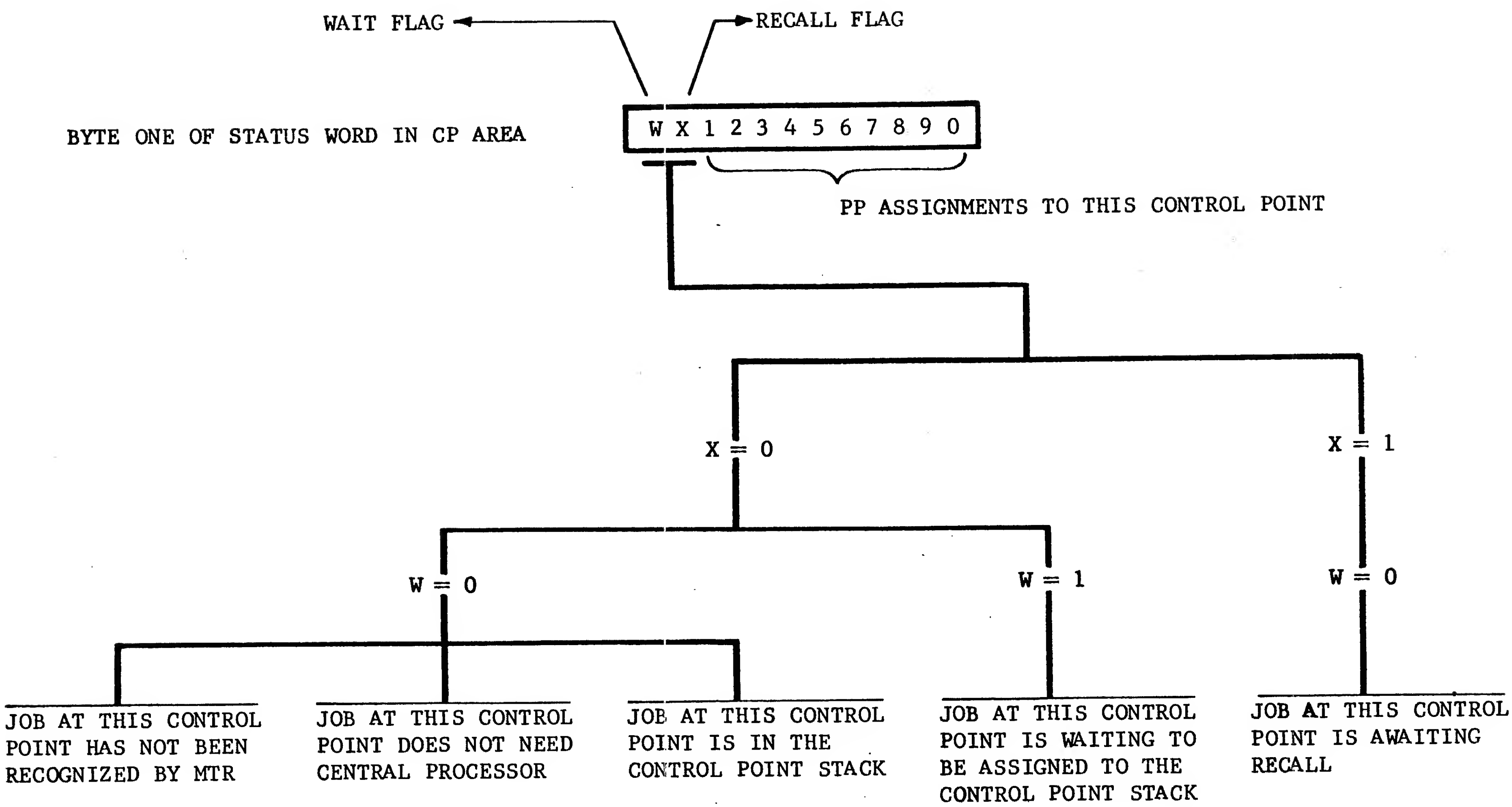



Figure 5

CP STATUS FLAGS

issues an exchange jump to interrupt the current program and initiate the new job. If the priority of the job currently being executed is higher than the priorities of any job which is in wait status (W flag set), then the routine leaves the flag set. This priority search is repeated on a periodic basis.

When a central processor program issues a recall request (by placing RCL in RA + 1), MTR processes this by interrupting this program and initiating the next program in the stack, and then setting the X flag in the control point area of the interrupted program. At an interval of time after the X flag is set, MTR will switch the control point from X status to W status and call the Search for CP Priority routine to re-initiate the job.

If the X flag is set, then, the job at the associated control point is awaiting recall. If the W flag is set, the job is waiting to enter the stack. The stack is always entered at the top: a job always enters the stack by taking control of the central processor. Note that the W flag and the X flag are never both set at the same time. If the W and X flags are both cleared, there are three possibilities:

- the job at the associated control point is in the stack
- the job at the associated control point does not require the central processor
- the job at the associated control point is inactive or has not yet been recognized by MTR

The interpretation of the X and W flag settings is charted in figure 5.

The status of each control point is examined by a MTR subroutine called Advance CPU Job Status. This routine is called each time MTR makes a pass through its master loop: however, unless 64 milliseconds or longer has passed since the routine was last entered, control is immediately returned to the master loop. This routine examines only one control point on each entry:

thus, a minimum interval of 7 x 64 = 448 milliseconds elapses between successive scans of the same control point. The Advance CPU Job Status subroutine and its relationship to the MTR master loop are illustrated in figure 6. (See page A-13 of the attached flow charts for a more detailed flow chart of this routine.)

Upon entering the routine (if the 64 ms. interval has elapsed) the pointer for the control point to be scanned is advanced. This pointer is maintained in location 26 of peripheral processor zero's memory. The X flag for the control point is then examined: if this flag is set, the W flag is set and the X flag cleared. The subroutine Search for CP Priority is then called to re-initiate the program. If this program's priority is higher than the priority of the program currently being executed, the running program will be interrupted and its control point address pushed down in the stack: the higher priority program will be initiated, its control point address placed at the top of the stack, and its W flag cleared.

After processing the central processor recall flag, the Advance CPU Job Status routine examines the PP recall word in location $25_8$ of the control point area. If this word is non-zero, a peripheral processor is assigned to complete execution of the recalled task.

The routine next examines byte one of location $20_8$ in the control point area: if this byte is non-zero, then the W flag or the X flag at this control point is set, and/or a peripheral processor is performing a task for the job at this control point. If this byte is non-zero, then, the job at this control point is active (although perhaps not in execution at the moment): the routine therefore exits back to the master loop.

If the storage move flag in byte three of location $20_8$ in the control point area is set, the storage assigned to this control point is to be or is being relocated. The routine exits back to the master loop, thus delaying further action until this relocation is completed.

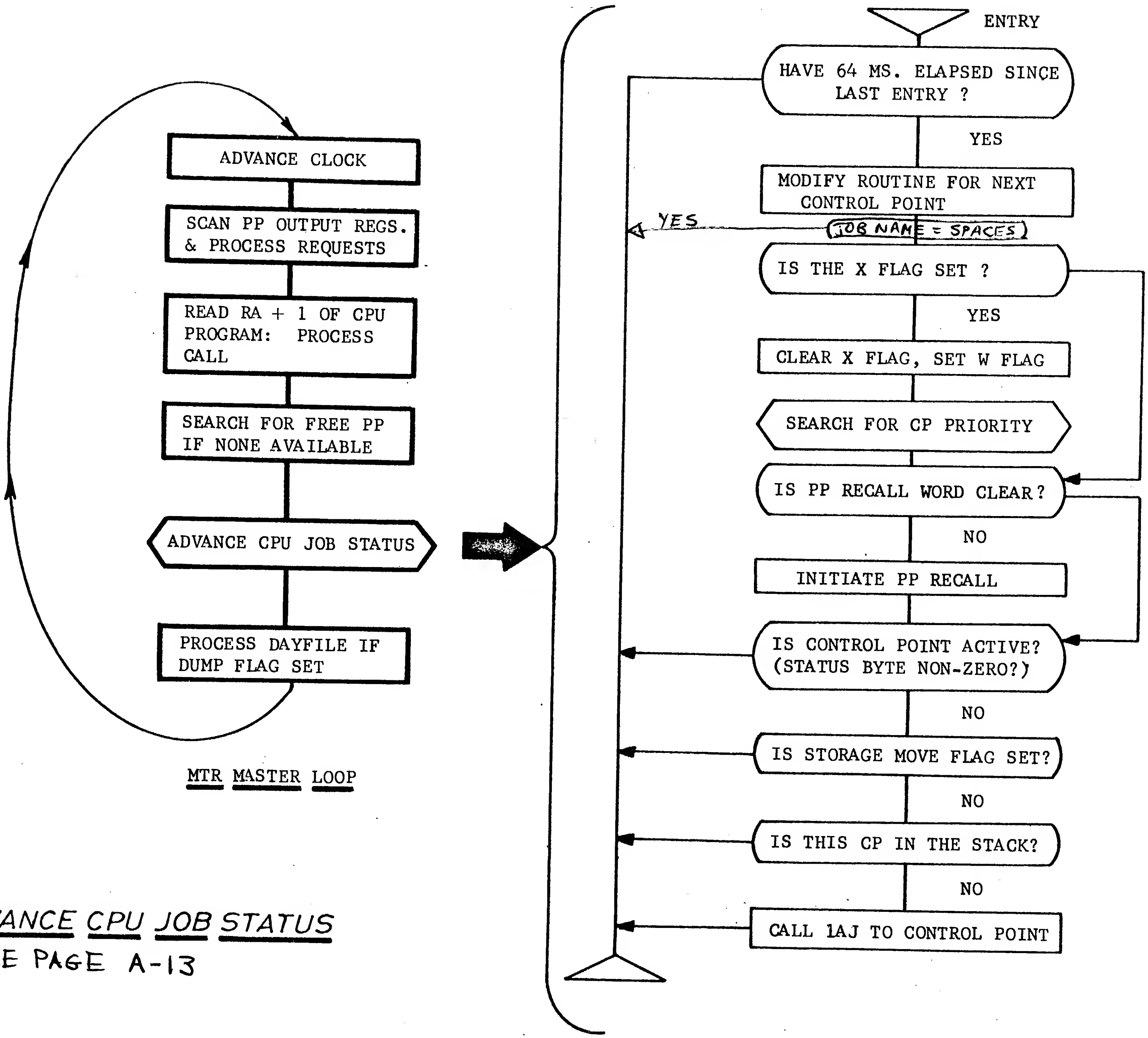



Figure 6

ADVANCE CPU JOB STATUS
SEE PAGE A-13

It is possible for byte one of the status word (control point area, location $20_8$) to be zero and for the program at that control point nevertheless to be active. For example, the non-executing programs in the stack will have zero status (i.e., byte one of the status word = 0), and the running program will also have zero status if it is not using a peripheral processor. Therefore, in addition to determining that the program has zero status, the routine must also determine if the job is in the control point stack before it can be ascertained that all activity associated with the control point has stopped. If it is found that this is the case, the routine assigns a peripheral processor to the control point and calls the 1AJ routine to that processor. The 1AJ routine promptly calls the statement translator, 2TS, to interpret the next control statement. (Note: although not shown in the simplified flow chart of figure 6, the Advance CPU Job Status routine also checks to see if the running program has exceeded its time limit; if so, the appropriate error flag is set.)

Now let us return for a moment to an earlier point in our discussion. After 1BJ has brought a job to its control point, the job name, time limit, priority, and field length have been set in the control point area. Also, byte one of the status word is zero. Further action involving the control point is initiated by the MTR subroutine Advance CPU Job Status. When this routine scans the control point to which the job was brought by 1BJ, it finds that the job has zero status (byte one of location $20_8 = 0$) and that the job's control point address is not listed in the stack. Thus, there is no activity at this control point. The Advance CPU Job Status routine therefore assigns a pool processor to this control point and calls 1AJ to that processor (by setting the name in the processor's Input Register). 1AJ in turn calls an overlay, 2TS, to process the next control statement from the control statement buffer. If this is a statement such as ASSIGN, RELEASE, COMMON, etc., the statement translator, 2TS, processes the control statement, moves the next control statement pointer (byte 5 of location $21_8$ in the control point area) to point

to the next control statement in the buffer, and releases the processor. This ends the activity at the control point temporarily: the Advance CPU Job Status routine will recognize this inactivity, and call 1AJ to advance the job at this control point again. This process repeats until a statement not recognized as a control point statement - a program card - is processed by 2TS. When 2TS processes a program card, it searches the FNT, the CLD, and the PLD, in that order, for the program. If the program is found in the FNT, 2TS proceeds to read the program from disk 0 into central memory beginning at location RA. Upon reaching the end of record (or upon detecting the end of the storage assigned to the job), 2TS sets the proper value of P in the exchange area, sets the field length in $A_0$, transfers the arguments from the program card to the program area beginning at RA + 2, and clears RA and RA + 1. The value of P is obtained by adding 3 to the number of arguments: the latter quantity is supplied by the low-order six bits of the second word in the program record. The field length is set in $A_0$ so that the program can determine the upper limit of its memory area. The remainder of the exchange area is cleared.

When 2TS has completed setting up the control point area and the program area, it requests the central processor for the job by sending function code 15 (Request Central Processor) to MTR. MTR sets the W flag in the control point area, and calls the Search for CP Priority subroutine to initiate the job. This subroutine will compare the priority of the new job with the priority of the running job, initiating the execution of the new job if it is higher in priority.

## MTR: EXCHANGE PACKAGE SWITCHING

As jobs are brought into the system or are recalled from X status, the running program may be interrupted to permit a higher priority program to take the central processor: when this occurs, an exchange jump is issued which results in the exchange package of the interrupted job being stored in

the control point area of the newly initiated job. Also, the stack is pushed down and the control point address of the newly initiated job is placed at the top of the stack. Each control point in the stack contains the exchange package for the control point immediately below it in the stack. An example will help to illustrate how this comes about. Assume that 1BJ routines at control points 3, 4, and 5 bring jobs C, B, and A, respectively, to their control points. Job C has a priority of 1, job B has a priority of 2, and job A has a priority of 3. At the time the jobs are loaded, the central processor is executing the idle program. Figure 7 illustrates a possible sequence of events involving these jobs:

1. 1BJ has brought jobs C, B, and A to control points 3, 4, and 5. The central processor is executing the idle program and thus the top of the stack contains zero - the address of the control point area for pseudo control point zero.

2. The Advance CPU Job Status routine has recognized the presence of the job at control point 3 and called 1AJ to advance the job. 1AJ's overlay, 2TS, has loaded the program into memory and, via a MTR request, requested that the job be executed. The MTR subroutine which processed this request called the Search for CP Priority routine. Since job C has a higher priority than the running program, the latter routine issued an exchange jump to job C which resulted in the idle program's exchange package being stored in control point three's exchange area, and then, after pushing down the stack, placed the address of control point three at the top of the stack.

3. When the process described above was performed for control point 4, the Search for CP Priority routine recognized that job B had a higher priority than job C. It therefore issued an exchange jump to start job B, thus storing C's exchange package

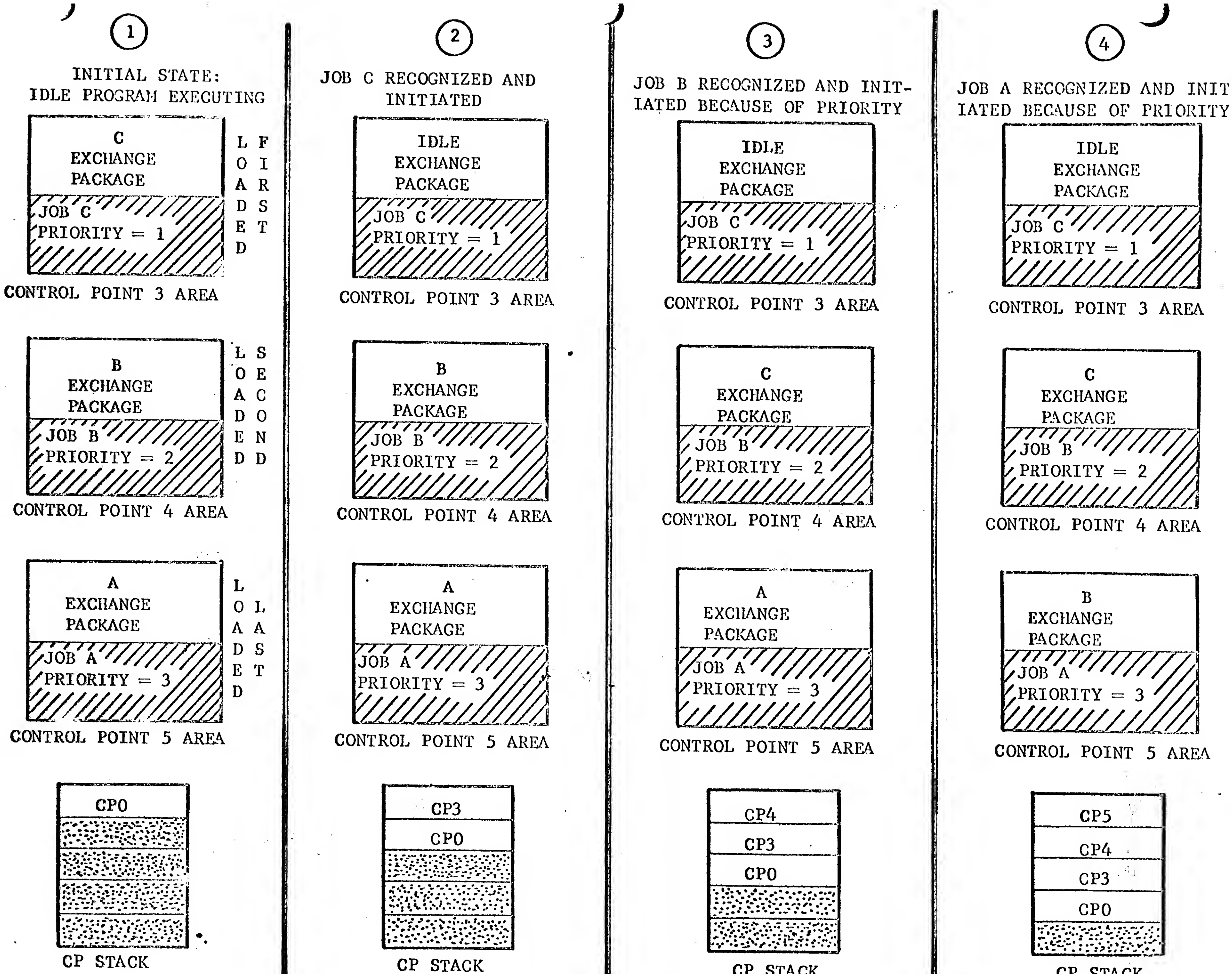

1
INITIAL STATE:
IDLE PROGRAM EXECUTING
C
EXCHANGE
PACKAGE
JOB C
PRIORITY = 1
CONTROL POINT 3 AREA
LOADED FIRST
B
EXCHANGE
PACKAGE
JOB B
PRIORITY = 2
CONTROL POINT 4 AREA
LOADED SECOND
A
EXCHANGE
PACKAGE
JOB A
PRIORITY = 3
CONTROL POINT 5 AREA
LOADED LAST
CP0
CP STACK
2
JOB C RECOGNIZED AND
INITIATED
IDLE
EXCHANGE
PACKAGE
JOB C
PRIORITY = 1
CONTROL POINT 3 AREA
B
EXCHANGE
PACKAGE
JOB B
PRIORITY = 2
CONTROL POINT 4 AREA
A
EXCHANGE
PACKAGE
JOB A
PRIORITY = 3
CONTROL POINT 5 AREA
CP3
CP0
CP STACK
3
JOB B RECOGNIZED AND INIT-
IATED BECAUSE OF PRIORITY
IDLE
EXCHANGE
PACKAGE
JOB C
PRIORITY = 1
CONTROL POINT 3 AREA
C
EXCHANGE
PACKAGE
JOB B
PRIORITY = 2
CONTROL POINT 4 AREA
A
EXCHANGE
PACKAGE
JOB A
PRIORITY = 3
CONTROL POINT 5 AREA
CP4
CP3
CP0
CP STACK
4
JOB A RECOGNIZED AND INIT-
IATED BECAUSE OF PRIORITY
IDLE
EXCHANGE
PACKAGE
JOB C
PRIORITY = 1
CONTROL POINT 3 AREA
C
EXCHANGE
PACKAGE
JOB B
PRIORITY = 2
CONTROL POINT 4 AREA
B
EXCHANGE
PACKAGE
JOB A
PRIORITY = 3
CONTROL POINT 5 AREA
CP5
CP4
CP3
CP0
CP STACK

Figure 7

in B's exchange area in control point four. The stack was pushed down and the address of control point 4 placed at the top of the stack.

4 The process is repeated again for control point 5: job A takes over the central processor, and the address of control point 5 is placed at the top of the stack after the stack has been pushed down.

Should job A complete execution or enter recall status, an exchange jump is issued in which the address specified for the exchange package is the address at the top of the stack. The stack is then pushed up. This would cause the exchange package for job A to be stored in control point five's exchange area: the stack and control point areas would then appear as shown in 3 .

The use of the central processor by a job may be suspended by means of a "DCP" keyboard entry to the DIS package assigned to the job's control point. When DIS encounters this entry, it transmits the control point number and function code 16 (Release Central Processor) to MTR. The MTR subroutine which processes this request determines whether or not the control point is in the stack. If the control point is not in the stack, the W and X flag bits are set to zero, and the routine exits. Although the job's control point is not in the stack and neither the W flag nor the X flag is now set, the Advance CPU Job Status routine will not consider this job inactive, since byte one of the status word is non-zero by virtue of the fact that a bit is set corresponding to the number of the peripheral processor containing the DIS package.

If the MTR subroutine which processes this request finds that the control point address of the job to be suspended is contained in the stack, it must push the control point address of the job up out of the stack and reorder the exchange packages so that the control point area of the suspended job con-

tains its own exchange package. To accomplish this, MTR exchanges the running program with the program immediately below it in the stack, and pushes up the stack. If the program exchanged (i.e., the former running program) was not that of the job to be suspended, MTR sets the W flag for this job and repeats the above process. When the job to be suspended has been exchanged, MTR calls the Search for CP Priority routine to reconstruct the stack. The W flag for the suspended job is not set.

Many of the DIS entries which modify program parameters utilize this function to halt the running program so that parameters can be changed.

To re-initiate execution of the suspended job, the DIS keyboard entry "RCP." is used. On detecting this entry, DIS sends function code 15 (Request Central Processor) to MTR. MTR then sets the W flag for this job and calls the Search for CP Priority routine to re-initiate execution.

Several MTR subroutines are required to push up the stack to extract a control point address: these routines call the Search for CP Priority routine to reconstruct the stack. The latter routine is the only routine which pushes down the stack and adds new entries to it.

Whenever a routine pushes the stack up or down, a copy of the stack is written in locations 56 and 57 of central memory resident for use by DSD. DSD uses an alphabetic code to indicate the position of a control point in the stack. The control point at the top of the stack (i.e., the running program) is displayed as having program status "A", the next control point in the stack is displayed as having program status "B", and so forth. The W and X flags are also displayed for control points not in the stack.

MTR: PP RECALL PROCESSING

When certain transient programs find they cannot immediately continue to perform their functions, they enter a process called PP Recall. Some of the instances where this takes place are:

- 1BJ - while waiting for storage to be assigned
- 1DJ - while waiting for an output file
- 1LJ - while waiting for a card reader to become ready

To enter PP Recall, a routine simply copies the contents of its Input Register in the PP Recall register for the control point (location $25_8$ of the control point area), requests MTR to release the processor, and exits to the resident idle loop.

The PP Recall register in the control point area is examined by the Advance CPU Job Status routine. When this routine finds that the contents of the PP Recall register are non-zero, it recalls the task by copying the contents of the PP Recall register into the Input Register of an available pool processor, clearing the PP Recall register, and assigning the processor to the control point (by setting the appropriate bit in byte one of the status word). The design of the transient programs is such that no internal modifications or special flags are required for recall: a recall entry is treated just like an initial entry.

The recall process is also utilized in the loading of peripheral processor programs. When the statement translator, 2TS, processes a program card, it first assumes that the program requested is a central processor program, and searches the FNT and the CLD for the program. If the program is not found in either the FNT or the CLD, the statement translator then assumes that the request is for a peripheral processor program, and so searches the PLD. If the routine is found in the PLD, the statement translator places the routine name and control point number in the PP Recall register for the control point.

MTR's Advance CPU Job Status routine treats this as if it were a recall entry. It assigns a processor to the control point and copies the PP Recall register into the processor's Input Register. The processor's resident program then proceeds to load the program from the disk library and execute it.

## MTR: NORMAL AND ABNORMAL JOB TERMINATION

In normal termination of a central processor job, the central processor program initiates this termination by writing "END" in RA + 1. When MTR detects this request during its master loop, it exchanges this program with the program at the control point below it in the stack, and pushes up the stack. If all peripheral processor activity associated with this control point has ceased, then byte one of the status word in the control point area will be zero. When the Advance CPU Job Status routine detects that this control point is inactive, it will call the 1AJ routine to the control point. If all control statements in the control statement buffer have been processed, 1AJ will wrap up the job.

When a job at a control point involves only peripheral processors and does not use the central processor, normal termination involves a process similiar to that described above. When a peripheral processor program completes execution, it requests MTR to release the processor (function code 12). MTR does this by clearing the processor's Input and Output Registers and clearing the appropriate bit in byte one of the control point's status word. When all processors associated with this control point have been released, the job will have zero status. This will be detected by MTR's Advance CPU Job Status routine, which will call 1AJ to the control point.

Abnormal termination of a job may be initiated by a central processor program, a peripheral processor program, or by MTR. Regardless of who initiates abnormal termination, the general procedure followed by MTR is to set an error flag in byte two of the status word at the control point and cause the job to assume zero status by clearing the W or X flag, releasing peripheral processor assignments, and/or pushing the job's control point out of the stack. When the Advance CPU Job Status routine detects that the job at this control point has zero status, it will call 1AJ to advance the job. 1AJ senses that the error flag is set and calls an overlay to process the

remaining control statements in the control statement buffer. This overlay, 2EF, searches the control statement buffer for an EXIT statement: if none is found, control is returned to 1AJ to wrap up the job. If an EXIT statement is found, the control statement immediately following it is picked up for translation and processing.

For certain of the error flag conditions, the 2EF overlay inserts (via a MTR request) an error message in the dayfile. Error messages for other flags are placed in the dayfile by the initiating routine.

The setting and processing of the various error flags is described below.

<u>Error Flag 1: Time Limit</u>. Byte four of location 22 in the control point area contains the time limit in octal minutes for the job. Bytes three and four of location 23 in the control point area contain the central processor running time in seconds for the job. Each time the Advance CPU Job Status routine is entered, the running time of the active central processor program is incremented. The running time (bytes 3 and 4 of location 23 in the control point area) is then compared with the time limit (byte 4 of location 22). If the time limit has been exceeded, a subroutine called Set Error Flag (page A-15 of attached flow charts) is called. This subroutine drops the job from the central processor either by clearing the W/X flag or by exchanging the program with the next one in the stack and pushing up the stack. It then sets the error flag bit in byte two of the control point's status word. In the case of a time limit error, the error message is later inserted by 1AJ's overlay, 2EF.

<u>Error Flag 2: Arithmetic Error</u>. On each pass through its master loop, MTR reads the central processor P register. If (P) is zero, it is assumed that an error exit due to an infinite/indefinite operand or bounds error has occurred. MTR then calls the Set Error Flag subroutine to drop the job from the central processor and set the

$2^2$ bit in byte two of the control point's status word. The error message is later inserted by 2EF.

Error Flag 3. PP Abort. There are several instances in which a peripheral processor program finds it necessary to abandon a task. Some of these are:

- a peripheral resident is unable to locate a package in the resident or peripheral libraries
- CIO's overlay, 2BP, finds an error in the buffer parameters specified in a call
- a parity error is encountered when backspacing (after three attempts)

In these instances, the peripheral processor program sends an error message to the dayfile and then requests MTR to abort the control point (function code 13). MTR releases the processor (thus clearing the corresponding bit in byte one of the status word) and then calls the Set ErrorFlag to set the $2^3$ bit in byte two of the status word and to drop the job from the central processor.

Error Flag 4: CPU Abort. When a central processor program finds it necessary to abort execution, it writes "ABT" in RA + 1. When MTR detects this during its master loop, it calls the Set Error Flag Subroutine to set the $2^4$ bit in byte two of the status word and to drop the central processor.

The central processor program may abort because of some computational condition, or because of a problem in the execution of a peripheral processor program associated with the job. For example, if a tape read operation encounters a parity error, after the third unsuccessful read it sets the $2^0$ bit in byte four of RA and pauses (function code 17: Pause for Storage Relocation). The central processor program presumably monitors this location: when it detects

that this bit is set, it must decide whether to abort execution or to ignore the error. To ignore the error, the central processor program clears this bit: the peripheral processor program will sense when the bit is cleared and proceed with its execution. To abort execution, the central processor program places "ABT" in RA + 1, which results in an error flag being set. The peripheral processor program senses this error flag and, when it finds that the error flag is set, it requests MTR to release the processor. In either case, the peripheral processor program will place a message in the dayfile.

Error Flag 5: PP Call Error. When MTR senses, during its master loop, that the contents of RA + 1 are non-zero, it calls a subroutine to process the request. If the contents of RA + 1 are not END, RCL, or ABT, then it is assumed that a peripheral program is being called. The subroutine checks the first character of the call to see if it is a letter. If it is, the request is issued to a free pool processor. If it is not, the Set Error Flag subroutine is called to set the $2^5$ bit in byte two of the control point's status word and to drop the central processor. The 2EF overlay later inserts an error message in the dayfile.

Error Flag 6: Operator Drop. When DSD detects the keyboard entry "n.DROP." (n = control point number), it transmits the control point number and function code 30 (Operator Drop) to MTR. MTR calls the Set Error Flag subroutine to set the $2^6$ bit in byte two of the control point's status word and to drop the central processor.

Error Flag 7: Track Limit. The number of half tracks on disk 0 requested by peripheral processor programs assigned to a control point is maintained in byte 3 of location 22 in the control point area. This quantity is incremented as tracks are requested and decremented as tracks are dropped. Each time a track is requested,

MTR checks to see if more than $777_8$ half tracks have been assigned to this control point: if so, the Set Error Flag subroutine is called to set the flag in byte 2 of the control point's status word and to drop the job from the central processor. The 2EF overlay later inserts the error message in the dayfile.

If a routine requests a half track assignment from MTR, and MTR, in searching the Track Reservation Table, reaches the end of the table before an available half track is found, a zero byte is returned to the requestor in byte one of the first word in the Message Buffer. The requestor then aborts the control point via an MTR request, resulting in the setting of error flag 3.

The error message "TRACK LIMIT" indicates that a control point has requested the assignment of more than $777_8$ half tracks on disk 0. The error message "DISK X TRACK LIMIT" indicates that disk X has overflowed.

## MTR: STORAGE ALLOCATION AND RELOCATION

The blocks of central memory storage assigned to the various control points always occupy positions in central memory relative to the number of the control point to which they are assigned. Thus, the storage assigned to control point 2 appears immediately above the storage assigned to control point 1, the storage assigned to control point 3 appears immediately above that assigned to control point 2, and so forth. As the jobs at control points request and release storage, the storage assigned to higher control points is relocated up or down so that no gaps of unassigned storage appear between the storage blocks of consecutive control points. All unassigned storage appears at the high end of memory.

Peripheral processor programs request storage from MTR via a Request Storage function (function code 10). Whenever 1BJ brings a job to a control

point, it requests MTR to assign the storage specified on the job card. In addition, many peripheral processor programs request storage for their own use, primarily for buffers. Thus, 1BJ requests $300_8$ words of storage to use as a buffer in reading the record containing the control statements, 1LJ requests $4000_8$ words of storage to use in buffering jobs from the card reader to the disk, and so forth.

The MTR Request Storage subroutine is shown on page A-6 of the attached flow charts. Upon entry, a storage move flag is set in location 51 of peripheral processor zero's memory. This flag is the Output Register address of the requesting processor. The difference between the amount of storage requested and the amount currently assigned to the control point is then computed. For example, when 1BJ requests that storage be assigned for the job to be loaded, MTR computes the difference between the requested storage (from the job card) and the storage already assigned to the control point (the $300_8$ locations used as a buffer). If the requested storage represents an increase, MTR ascertains if there is enough unassigned memory available to provide room for the increase. It does this by subtracting (RA + FL) for control point 7 from $400000_8$ to determine the amount of unassigned storage. The storage increase requested is then compared with the amount of unassigned storage. If there is insufficient unassigned storage available to meet the request, MTR clears the requesting processor's Output Register, clears the storage move flag in location 50, and exits. The requesting routine senses if the storage requested has been assigned by reading the value of FL in byte 5 of location 20 in its control point area and comparing this value with the amount of storage requested.

If there is room for the storage increase, or if the request represents a decrease, MTR sets a storage move flag in each control point above the requesting control point. This flag is the $2^0$ bit in byte 3 of location 20 in the control point area. Thus, if the requesting processor is assigned to control point 4, storage move flags would be set in control points 5, 6, and 7.

After setting the storage move flags, MTR determines if there is any peripheral processor activity at the flagged control points. It does this by first examining byte one of the control point status word. If bits 2 - 9 of this byte are cleared, then there is no peripheral processor activity at this control point. If one of these bits is set, then MTR reads the Output Register of the corresponding processor. If this Output Register contains anything but a 17 function code, MTR exits from the Request Storage routine. Only when all control points whose storage is to be relocated either have no peripheral processor assignments or have paused for storage relocation by issuing a 17 function code does MTR proceed to relocate storage.

To relocate storage, MTR sets up the exchange package for the storage move program with the parameters required to effect the relocation. This exchange package begins at location $2000_8$ of the central memory resident. MTR then proceeds to push up the stack, exchanging each program in turn and setting the W flag for the control point, until control point 0 is at the top of the stack. The storage move program is then exchanged for the idle program. When the storage move program completes execution, it stops with P = 0. The Request Storage subroutine monitors P and, when it becomes zero, exchanges the idle program for the storage move program. The RA value in each of the flagged control point areas (both in byte five of location 20 and in the exchange package) is then updated by adding the increase to the original value, and the storage move flags cleared. The FL value is then set in the exchange package and status word of the control point area for the requesting processor. Finally, the Search for CP Priority routine is called to reconstruct the stack, the storage move flag in location 50 is cleared, and the Output Register of the requesting processor is cleared.

Many peripheral processor programs, such as 1BJ, 1DJ, and 1LJ, enter PP Recall if a storage request cannot be immediately satisfied because of lack of space. In this case (insufficient unassigned storage), the Request Storage

routine clears the requesting processor's Output Register and the storage move flag in location 50 prior to exiting. Even though sufficient unassigned storage is available, storage relocation can be initiated only when all peripheral processor activity has ceased for the control points whose storage is to be relocated. Should one or more control points have active peripheral processors, the Request Storage routine exits without clearing the requesting processor's Output Register. This does two things; it inhibits the requesting processor's resident from exiting the Process Request subroutine, and it causes MTR to re-enter the Request Storage subroutine on every pass through its master loop, since the request remains in the processor's Output Register. Requests for storage from other processors are ignored while this request is in process. Effectively, then, MTR checks, on every pass through its master loop, the peripheral processor activity at the flagged control points to see if relocation can be initiated.

Cessation of peripheral processor activity may come about because a processor has completed its assigned task and requested MTR to release it (in which case it may immediately be assigned to another task), or because the processor has paused by sending function code 17, Pause for Storage Relocation, to MTR. When MTR detects this function request, it examines the storage move flag in the associated control point area: if set, MTR returns to its master loop without clearing the processor's Output Register, thus effectively stopping the processor. When all processors assigned to the flagged control points have paused, storage relocation can begin.

The peripheral processor residents all pause for storage relocation immediately upon recognizing a request in their Input Registers. In addition, many peripheral processor programs pause for storage relocation when delayed in their execution. For example, tape drivers pause for storage relocation when the tape unit is not ready or when a tape error occurs.

The READ package (1LJ and its overlays) pauses for storage relocation only when the card reader is not ready. Similiarly, the PRINT package (1DJ and its overlays) pauses for storage relocation only when the line printer is not ready. It is therefore important that these packages should be assigned to control points one and two. Should they be assigned to higher control points, they could hold up the allocation of storage for jobs at lower control points for considerable periods of time.

The coding for the storage move program is shown on page B-1, together with the 6600 central processor timing for the loop used in moving storage up. Storage relocation requires approximately 7.2 microseconds for an increase of $10{,}000_8$ words.

MTR: TIME ACCOUNTING

Location 30 of the central memory resident contains the system time in hours, minutes, and seconds. This location may be initialized to real time via the DSD keyboard entry "TIME". If not initialized, this location reflects the elapsed time since the system was loaded. This time is updated by the Advance Clock subroutine. This routine is shown on page A-2 of the attached flow charts.

In addition to maintaining the system time in location 30 of central memory resident, the subroutine also maintains a current second count and a current millisecond count in locations 67 and 66, respectively, of PP0's memory. Upon entering the Advance Clock subroutine, MTR reads the real time clock on channel 14 and extracts the high-order two bits. These two bits are interpreted as follows:

high-order bits = 00: real time clock has advanced 0 milliseconds
high-order bits = 01: real time clock has advanced 1 millisecond
high-order bits = 10: real time clock has advanced 2 milliseconds
high-order bits = 11: real time clock has advanced 3 milliseconds
high-order bits = 00: real time clock has advanced 4 milliseconds
. . . . . . .        . . . . . . . . . . . . . . . .

These two bits are compared with the clock phase in location 65 of PPO's memory. This clock phase is the value of these two bits on the last entry to the subroutine: if these two bits are unequal to the clock phase, then a millisecond has elapsed since the last entry, and so the millisecond count in location 66 is incremented.

As the real time clock runs through a full period, the millisecond count is advanced by 4. Actually, a full period represents 4.096 milliseconds: therefore, rather than waiting for the millisecond count to reach 1000 before advancing the second count, MTR advances the second count when the millisecond count reaches 976. This represents 244 full periods of the real time clock, or an actual elapsed time of 999+ milliseconds.

If the second count was not advanced, the real time clock is read again before exiting to determine if a millisecond advance has taken place while the computations described above were taking place. If no advance has occurred, the subroutine is exited. If the second count was advanced, MTR reads the system time from location 30 of central memory resident, updates it, and writes it back in central memory. The real time clock is then read again to determine if an advance has occurred: if no advance has occurred, the subroutine is exited.

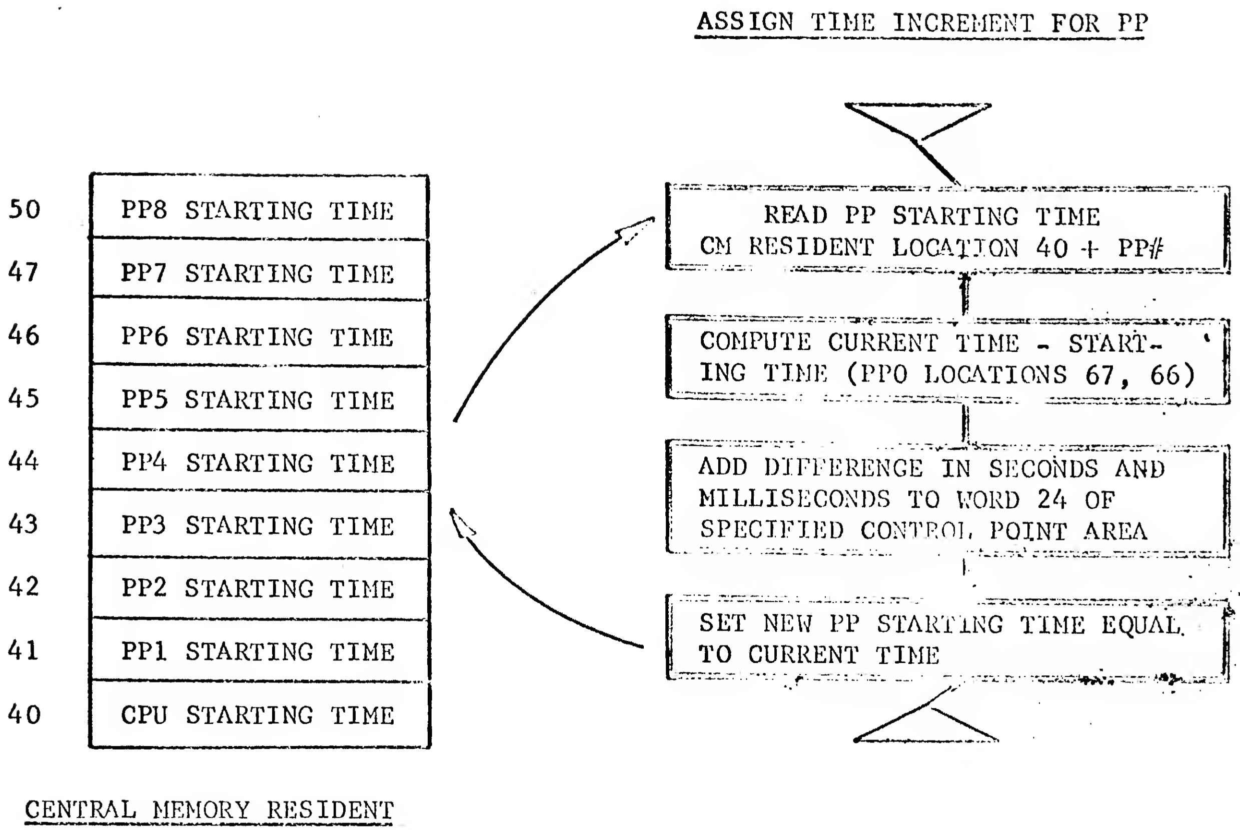
In order for the system time to be properly maintained, MTR must, on the average, enter this subroutine every millisecond. Therefore, entry to this subroutine is made from several points in MTR. The MTR routines which call the Advance Clock subroutine are:

- MTR master loop
- Process PP Message Routine
- Request Storage (function 10)
- Release Central Processor (function 16)
- Request Exit Mode (function 25)
- Set Error Flag Routine

The system time in location 30 of central memory resident is inserted by MTR in each dayfile message and is displayed by DSD. This time is not, however, used in computing time charges to control points.

Location $23_8$ in the control point area holds the central processor time charged to the job at the control point, while location $24_8$ contains the peripheral processor time charged to the job. These times are maintained in seconds and milliseconds, and are entered in the dayfile by 1AJ upon completion of the job. Peripheral processor time charges are accumulated by the Assign Time Increment for PP subroutine. This subroutine maintains a starting time for each pool processor in central memory locations 41 - $50_8$. This starting time represents the time at which the peripheral processor most recently became idle or active and is maintained in seconds and milliseconds. The Assign Time Increment for PP subroutine is illustrated in figure 8. This subroutine is entered with the number of the pool processor and with the control point address. On entry, the starting time for this processor is read from central memory (location 408 + PP number) and subtracted from the current time in seconds and milliseconds maintained in locations 67 and 66 of PP0's memory. The difference is added to the contents of word $24_8$ in the specified control point area. The starting time for the processor is then reset to the current time in seconds and milliseconds.

When MTR assigns a pool processor to a task, it enters this subroutine with the number of the processor and with the address of control point zero. The difference between the starting time and the current time is the length of time which the processor has been idle. The new starting time represents the time at which the processor began execution of the task assigned to it by MTR. On completion of the task, MTR agains enters the subroutine - this time with the number of the processor and the address of the control point to which the processor was assigned. The difference between the current time and the starting




- ON ASSIGNING A POOL PROCESSOR TO A CONTROL POINT, THE ROUTINE IS ENTERED WITH THE PP NUMBER AND THE ADDRESS OF CONTROL POINT 0
- ON RELEASE OF A POOL PROCESSOR, THE ROUTINE IS ENTERED WITH THE PP NUMBER AND ADDRESS OF THE CONTROL POINT TO WHICH THE PP WAS ASSIGNED

Figure 8

PP TIME ACCOUNTING

time is the length of time the processor was assigned to the control point: the new starting time is the time at which the processor becomes idle. This subroutine accumulates all PP usage times for a job in word 24 of the control point area. All the idle time for the pool processors is accumulated is control point zero's area - location 24 of central memory resident.

The Assign Time Increment for PP subroutine is called by the following MTR routines:

- Assign PP Time to CP (function 4)
- Release PPU (function 12)
- Abort Control Point (function 13)
- Request PPU (function 20)
- Process PP Call
- Advance CPU Job Status

Routines processing jobs not associated with the control point, such as the READ and PRINT packages, must handle their own time charges. When these routines begin processing a file, they send function request 4 to MTR. MTR assigns the idle time to control point zero and sets a new starting time for the processor in which the routine resides. The routines then sets location 24 in the control point area to zero. When processing of the file is completed, these routines again send function request 4 to MTR, and MTR computes the processing time and stores it in location 24 of the control point area. The routines then read this time from the control point area, convert it to decimal, and write it in the dayfile via an MTR request. MTR inserts the job name in the dayfile message: it is to provide this job name that these routines change the job name in the control point area from READ or PRINT to the file name when processing of the file is initiated.

Time charges for the central processor are accumulated in a similiar manner. There is, however, one exception: since central processor programs have a time limit, the central processor time charges to a control point are

advanced every second. Location 40 in central memory resident contains the central processor starting time. At intervals of one second or less, this time is read and subtracted from the current time in seconds and milliseconds maintained by MTR in locations 67 and 66 of PPO's memory. The difference is added to the contents of word 23 in the specified control point area, and a new starting time is set in location 40 in central memory. (Note: the control point area is cleared at dead start time and whenever 1AJ drops a job from a control point.) Central processor time charges are updated by the Advance CPU Job Status routine, and whenever the control point stack is pushed up or down (i.e., whenever the running program is exchanged).

The Advance CPU Job Status routine maintains a Last Second count in location 63 of PPO's memory. Each time the Advance CPU Job Status routine is entered, the Last Second count is compared with the current second count in location 67 of PPO's memory. If these two quantities are not equal, the Last Second count is updated, the central processor time charges are accumulated for the job currently using the central processor, and a test is made to determine if the time limit has been exceeded.

## MTR: THE DAYFILE

The dayfile is a combination of a time accounting medium and a job log. The contents of the dayfile include:

- all control cards
- all diagnostic messages
- job loading times, job execution times (both for the central processor and the peripheral processors), and job printing times
- messages to the operator

The dayfile is maintained as a COMMON file on the disk. In addition, a number of the most recent dayfile entries are displayed on the console by DSD. At the

end of a job, all dayfile entries for that job are printed as part of that job's output.

Messages are entered in the dayfile by peripheral processors via a request to MTR. A central processor program may enter a message in the dayfile by calling the MSG peripheral package. A peripheral processor program initiates the entry of a message in the dayfile by placing the message in its Message Buffer and then placing function code 1 (Process Dayfile Message) in byte one of its Output Register. The message may be up to six central memory words in length and is terminated by a zero byte in byte five of the last word of the message.

When MTR processes this request, it first checks the dayfile dump flag in location 64 of PP0's memory. If this flag is set, then dumping of the dayfile to the disk is in process, and so MTR returns to its master loop, delaying the processing of this message to later. If the dayfile dump flag is not set (i.e., location 64 contains zero), MTR proceeds with the processing of the message. The contents of the Message Buffer are copied into words 30 - 35 of the control point area for the control point to which the requesting processor is assigned. These locations (words 30 - 35), together with the control point status, the next control statement, and the exchange area, are displayed on the console by the DIS "B" display.

The dayfile message, together with the system time and the job name from the control point to which the requesting processor is assigned, is placed in the dayfile buffer. The dayfile buffer (see figure 9) is an area of central memory resident used to buffer dayfile messages to the disk. Its starting (i.e., FIRST) address and LIMIT address (last entry address plus one) are specified by bytes one and four, respectively, of the dayfile buffer (DFB) pointer word in central memory location 3. Bytes two and three of the DFB pointer contain the IN and OUT addresses for the buffer. In inserting the

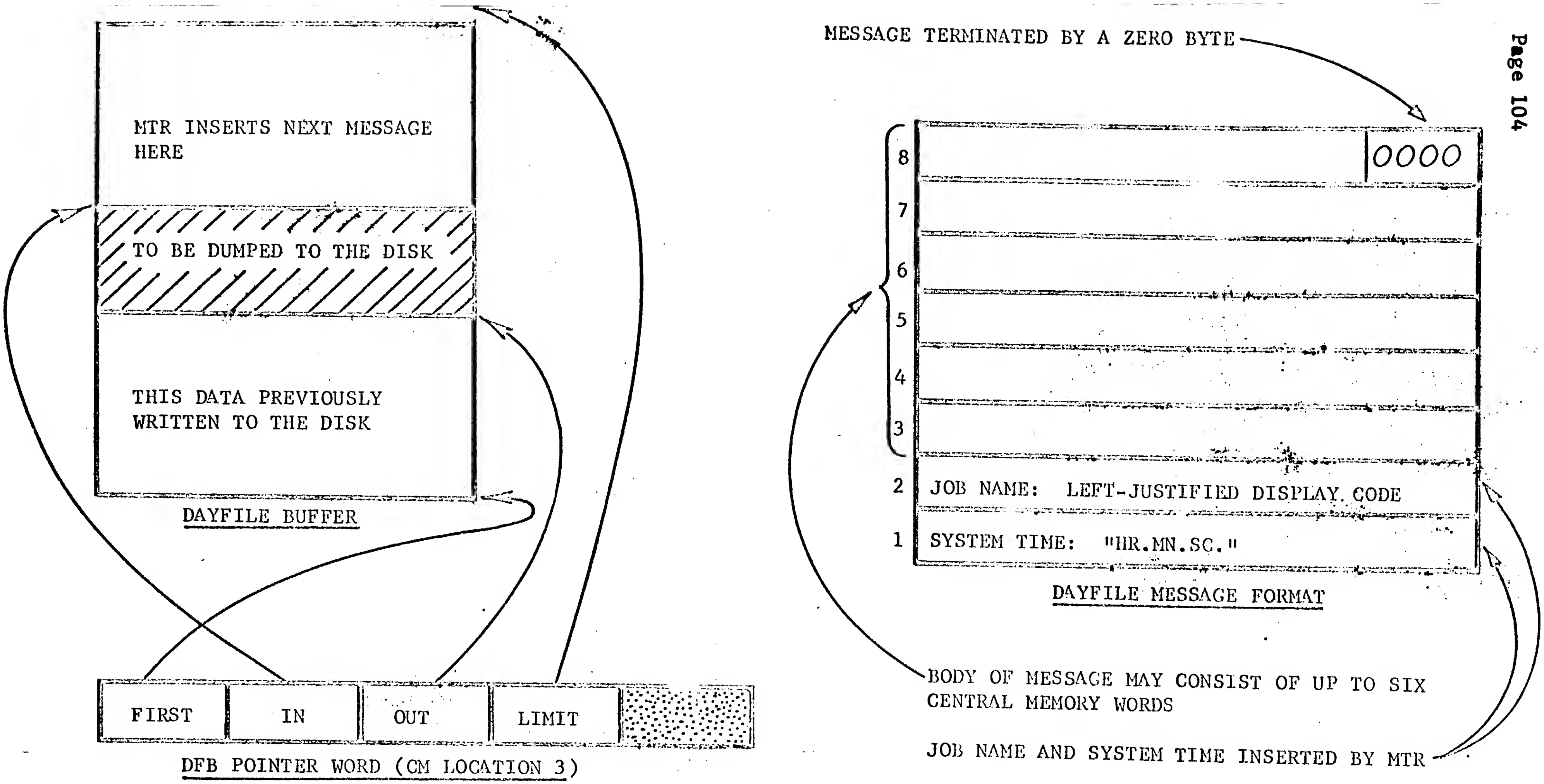



Figure 9

THE DAYFILE

message in the dayfile buffer, MTR first copies the system time from location 30 of central memory resident into the buffer. Next, MTR reads the job name from word 21 of the requestor's control point area and copies it into the dayfile buffer. In doing so, MTR changes zeroes in the job name to blanks, and inserts a period at the end of the job name. Next, MTR copies the body of the message from the requestor's Message Buffer into the dayfile buffer, copying word after word until a word ending with a zero byte (byte five) is copied.

Within the dayfile buffer, a message comprises three to eight words: one word contains the system time, one word contains the job name, and one to six words contain the body of the message. When the entire message has been copied into the dayfile buffer, MTR increments the dayfile message count in byte two of word 22 in the requestor's control point area. Although this count is incremented each time a message is entered in the dayfile, it is tested against a limit only by the peripheral package MSG.

As MTR enters each word in the dayfile buffer, it advances the IN address and compares it with the LIMIT address. When IN = LIMIT, MTR resets IN to the value of FIRST. After the message has been entered in the dayfile, MTR compares the IN and OUT addresses to determine if the dayfile buffer contains a full sector of data: if it does, the dayfile dump flag is set to initiate the dumping of this data to the disk. MTR dumps the dayfile to the disk in a series of phases: after each phase has been executed, MTR returns to its master loop to process requests from the central processor or from peripheral processors. In this manner, MTR avoids being tied up in a disk operation for a prolonged period of time. The dayfile dump flag, when set, contains the address of the subroutine to be called to perform the next phase of dumping.

Although the nominal size of the dayfile buffer is $1000_8$ words, dumping is initiated whenever messages totalling $100_8$ words have accumulated. From

the standpoint of buffering messages to the disk, the dayfile buffer need be be no longer than $107_8$ words (since it is possible that entry of the last message increased the total to over 100 words), since no entries can be made while dumping is in process. By increasing the buffer size to several times its minimum requirements, however, the size of the dayfile display on the console is increased.

The six subroutines corresponding to the dayfile dump phases are shown on pages A-16, A-17, and A-18 of the attached flow charts. These subroutines are described below.

Phase 1. In phase one, MTR requests channel 0 and sets the dump flag to the address of the phase 2 subroutine. It is interesting to note that in this case MTR transmits a request to itself: the channel number and the appropriate function number are placed in PP0's Output Register to be processed by MTR when it returns to its master loop.

Phase 2. One entering phase two, MTR reads its Output Register to determine if its reservation request has been accepted. If the channel has been reserved for MTR, then positioning is initiated. All other disk users maintain the current half track address for a file in the FST entry for that file. Although MTR sets the Beginning Half Track byte in the FST entry, it does not update the Current Half Track byte as sections of the dayfile are written to the disk. Instead, the current half track address is maintained by modifying the appropriate instructions within the dumping subroutines. To position the disk, MTR uses the Position Disk subroutine in peripheral processor resident. After initiating repositioning, the dump flag is set to the address of the phase 3 subroutine.

Phase 3. In phase three, MTR writes the full sector in the dayfile buffer and a record mark to the disk. Since this write is directed

to a specified sector, it is conceivable that up to 66 milliseconds could elapse between the time at which this subroutine was entered and the time at which this sector came under the heads. In order to avoid this delay, MTR issues a status request to obtain the number of the sector currently passing under the heads and, unless the disk is positioned two sectors before the desired sector, MTR returns to its master loop. A sector may pass under the heads in as little as 490 microseconds. The minimum time required for MTR to make a pass through its master loop is approximately 150 microseconds (assuming an active CPU program but no request processing required) and may be several times longer. It is not impossible, then, that a revolution or more may be required before the desired coincidence is found.

Once coincidence has been obtained, MTR writes the full sector from the dayfile buffer to the disk, and advances the buffer's OUT address accordingly. If this sector was the last sector on this half track, the subroutine coding is modified for the spare half track. (MTR maintains a spare half track for the dayfile: this is picked up in the phase four subroutine whenever required.)

It is probable that the message which completed a full sector in the dayfile buffer resulted in the buffer's containing something more than $100_8$ words of data. If so, MTR will include these extra words, which are part of the last message entered in the dayfile buffer, in the short sector written as an end-of-record after each full sector is written. (The dayfile is a single logical record on the disk and is not terminated by a file mark sector.)

The phase two subroutine is called again to position the disk, and the short sector is then written by the phase three subroutine.

Although this sector may include a few words of data from the buffer, the OUT pointer is not advanced to reflect the transfer of these words: also, the coding is not modified to reflect the writing of this sector. The next full sector written to the disk will also include these few words and will be written over this end-of-record sector.

After the end-of-record sector has been written, MTR constructs a release channel reservation request by placing the channel number and the appropriate function code in PPO's Output Register. It then sets the dump flag to either the address of the phase four subroutine (if another spare half track is required) or the phase six subroutine (if the spare half track was not used during this dump operation).

Phases 4, 5, 6. The phase four subroutine requests a spare half track from MTR and sets the dump flag to the address of the phase five subroutine. The phase five subroutine stores the spare half track number and clears the dump flag.

If it was not necessary to pick up the spare half track, the phase three subroutine sets the dump flag for phase six. The phase six subroutine clears the dump flag.

The dayfile buffer is dumped whenever messages totalling a full sector have accumulated. It is also dumped, even though a full sector has not been accumulated, at the end of each job. As part of a job's output, all dayfile messages for that job are printed. In order to simplify searching of the dayfile for the job's messages, the dayfile is dumped to the disk so that only the disk has to be searched. The PRINT package (1DJ and its overlays) and the DUMP package (1TD and its overlays) initiate this dumping by sending function request 11, Complete Dayfile, to MTR: MTR in turn sets the dayfile dump flag to phase one.

ADDITIONAL NOTES ON MTR'S ADVANCE CPU JOB STATUS ROUTINE

The Advance CPU Job Status routine performs a cyclic scan of control points, and initiates the processing of the next control statement (via 1AJ and its overlays) if the job at the control point being scanned is inactive. The interval between successive scans of the same control point is on the order of 450 milliseconds. When the Advance CPU Job Status routine determines that the job at the control point being scanned is inactive (i.e., the control point is not in the stack, the W or X flag is not set, and no peripheral processors are assigned), it calls 1AJ to advance the job. 1AJ in turn calls an overlay, 2TS, to process the next control statement. 2TS processes the control statement and releases the peripheral processor. Processing of the next control statement is initiated when the Advance CPU Job Status routine scans the control point and finds it inactive once again. Thus, if a job contained two ASSIGN cards, for example, the minimum interval between the initialization of processing for these control statements would be 448 milliseconds.

At first glance it would seem that an increase in throughput time could be achieved by reducing the interval between successive scans and/or modifying 2TS so that it would loop after processing one control statement to initiate processing of the subsequent control statement (unless a program card was encountered). Consider, however, that the most often used control statements (with the exception of JOB and PROGRAM cards, which are not pertinent to this discussion) are the ASSIGN, COMMON, and REQUEST cards. Let us briefly consider these control statements in terms of the delays which may be encountered in their processing.

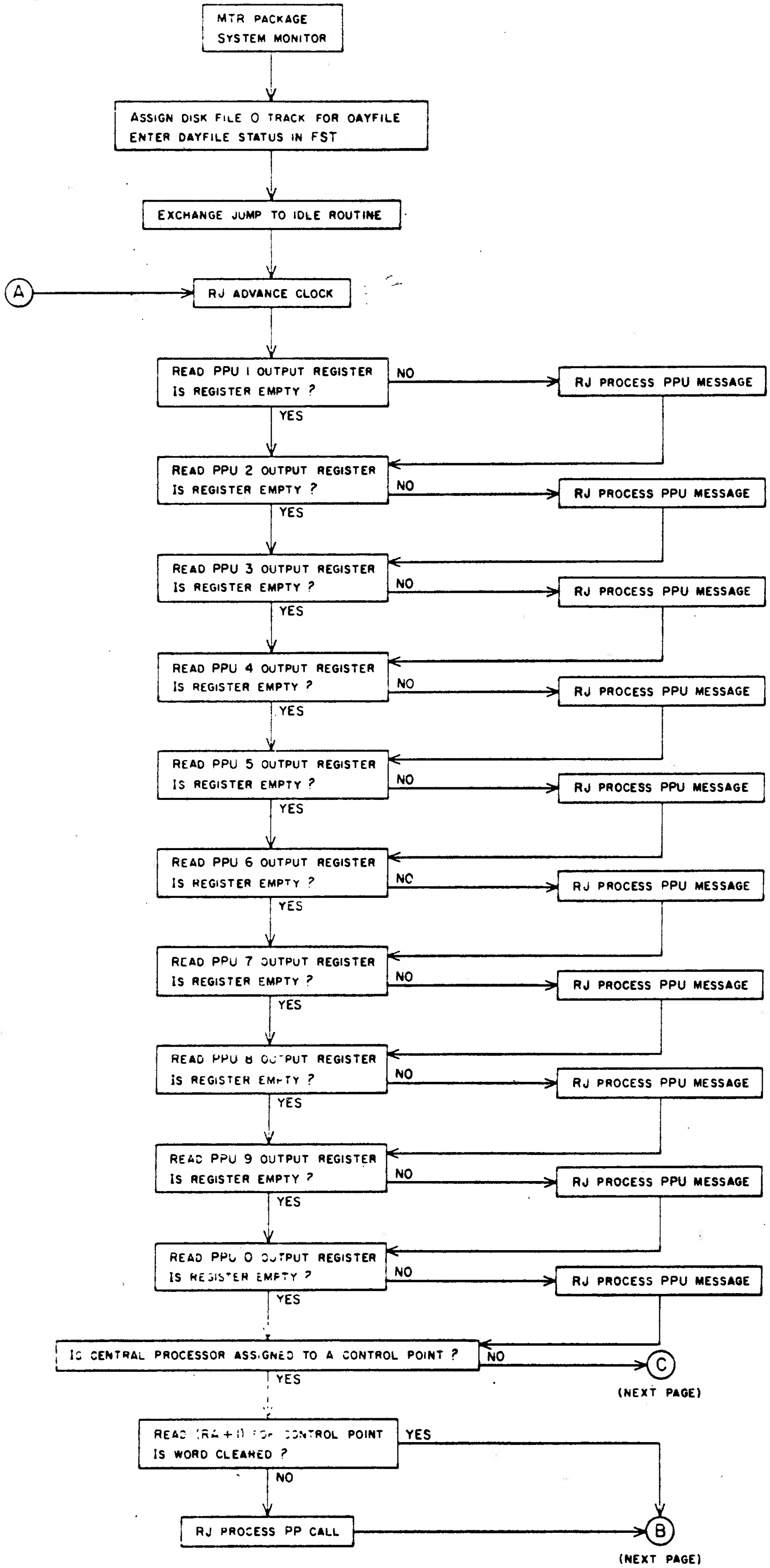
ASSIGN: the processing of the ASSIGN statement may be delayed if (1) operator action is required to supply an equipment number or (2) there is no space in the FNT to enter the file.

COMMON: the processing of the COMMON statement may be delayed if (1) the specified equipment is not available or (2) the file is being used by a job at some other control point.

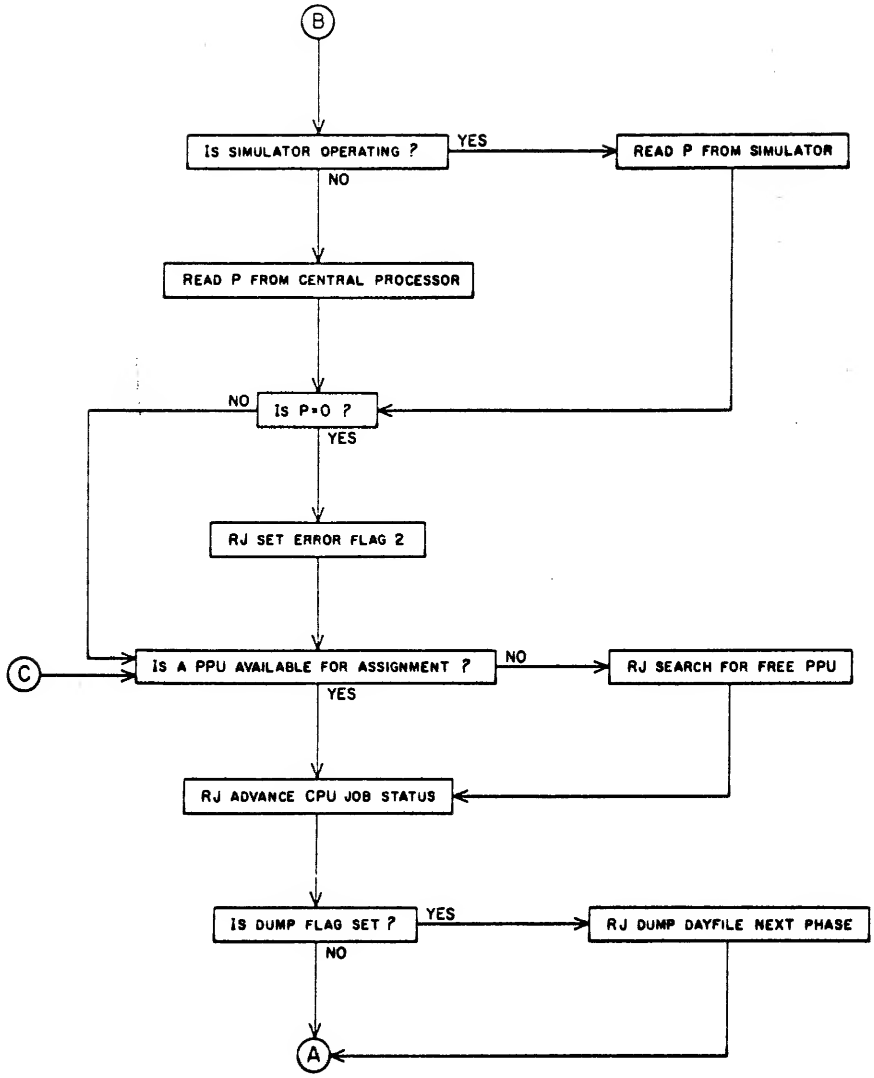
REQUEST: the processing of the REQUEST statement will be delayed (1) while the operator assigns an equipment and may be delayed (2) if there is no space in the FNT to enter the file.

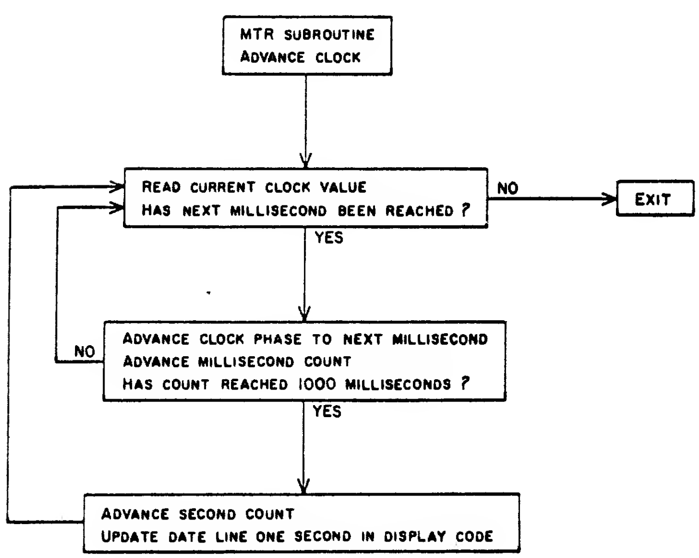
The delay most often encountered is the time spent in awaiting operator equipment assignment, which is always required by the REQUEST card and may be required by the ASSIGN card. In any event, when 2TS encounters a delay of the type described it releases the peripheral processor. When the Advance CPU Job Status routine scans the control point again, 1AJ will be called once again and its overlay, 2TS, will attempt to process the control statement again. Thus, although the time required for the operator to perform the assignment is measured in seconds (at best), the peripheral processor time is measured in microseconds. The net effect of this portion of the Advance CPU Job Status routine and 2TS overlay design, then, is to greatly increase the available peripheral processor time over that which would be achieved by designing 2TS in the manner described above. Furthermore, in view of the operator's reaction time, there is little point in reducing the interval between successive scans of a control point by the Advance CPU Job Status routine.

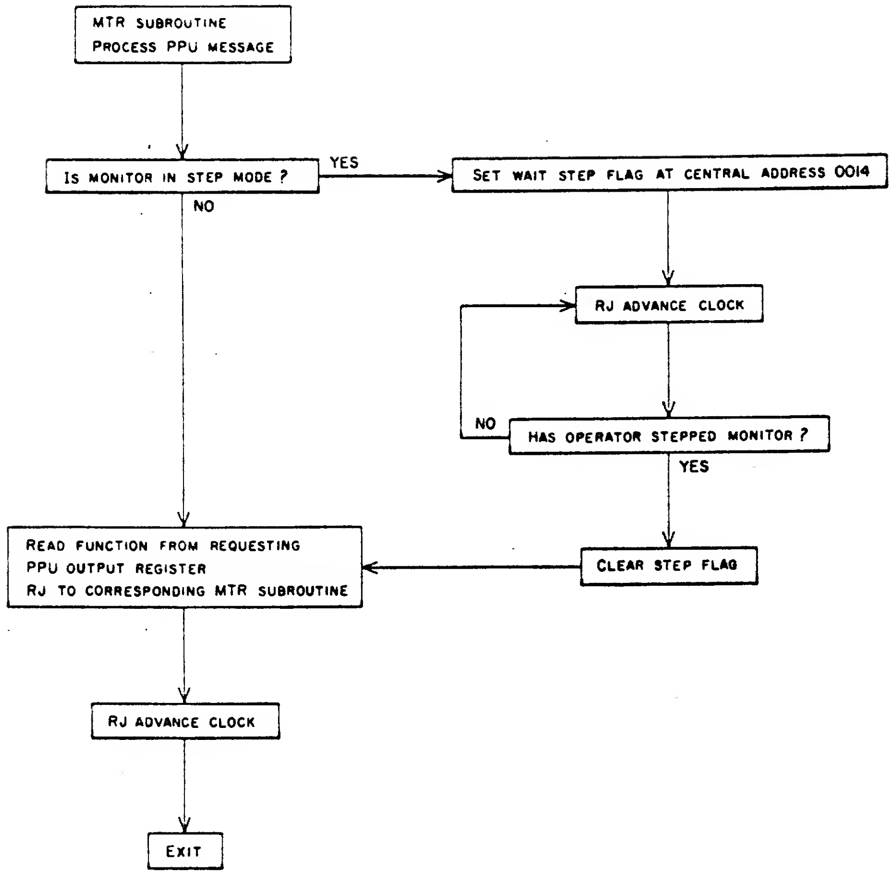
When the Advance CPU Job Status routine finds that the CPU program at the control being scanned is in recall status, it clears the X flag and sets the W flag for the job, and calls the Search for CP Priority subroutine to re-initiate execution. The net effect of this is to bring a job back from recall every (neglecting priority considerations) 450 milli-seconds. Since CIO will re-initiate execution of the job when the I/O operation is completed, this periodic return from recall may seem redundant. However, there is a purpose served (in addition to real-time and "quantum" type uses) by this return. When a I/O driver encounters a parity error, it sets a pause bit in RA ($2^{12}$ bit) and then loops, reading in turn the pause bit and the error flag byte until either the pause bit is cleared (in which case the error is ignored) or the error flag is set. The operator, on detecting the error message sent by the I/O driver, may clear the pause bit (n.GO entry to DSD) or set an error flag (n.DROP entry to DSD). The central processor program, however, on this periodic return from recall, may also sense the pause bit and take its own error action. The program may decide to accept the faulty data, in which case it would clear the pause bit and thus permit the I/O driver to proceed, or it may decide to abort, in which case it would set an error flag by writing ABT in RA+1, which would permit the I/O driver to exit.

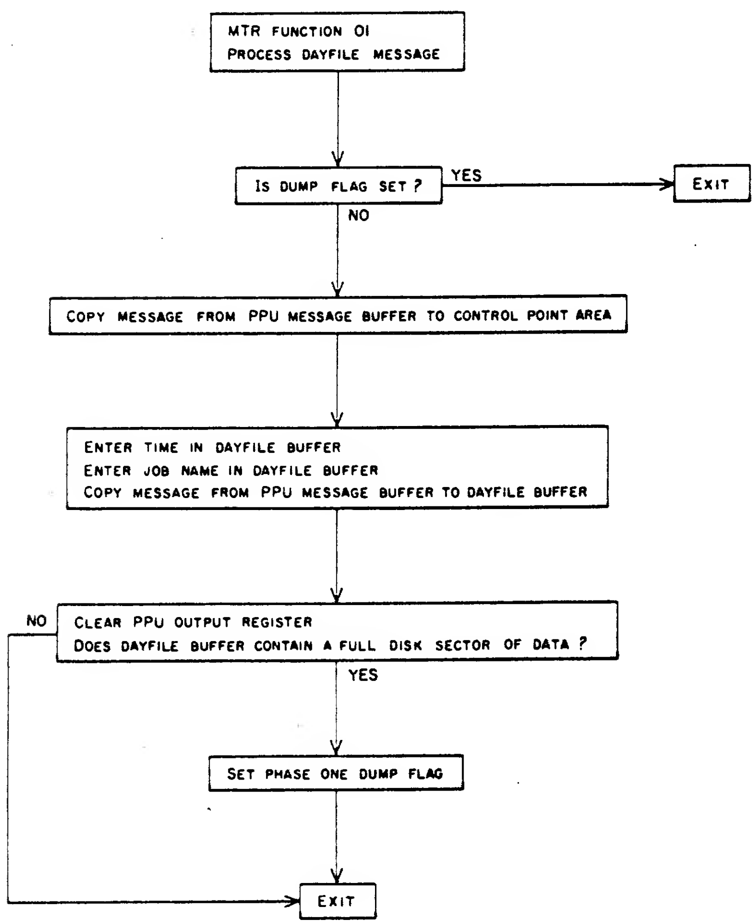

MTR PACKAGE
SYSTEM MONITOR
ASSIGN DISK FILE O TRACK FOR DAYFILE
ENTER DAYFILE STATUS IN FST
EXCHANGE JUMP TO IDLE ROUTINE
A
RJ ADVANCE CLOCK
READ PPU 1 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 2 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 3 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 4 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 5 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 6 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 7 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 8 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 9 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 0 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
IS CENTRAL PROCESSOR ASSIGNED TO A CONTROL POINT ?
NO
C
(NEXT PAGE)
YES
READ (RA+1) FOR CONTROL POINT
IS WORD CLEARED ?
YES
NO
RJ PROCESS PP CALL
B
(NEXT PAGE)

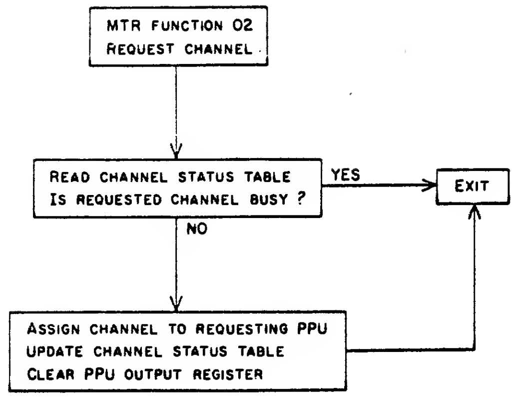
(MTR PACKAGE CONTINUED)

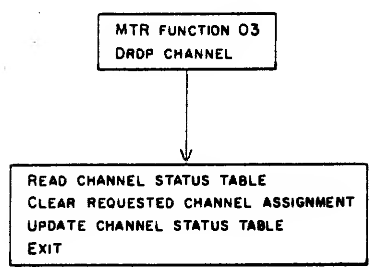


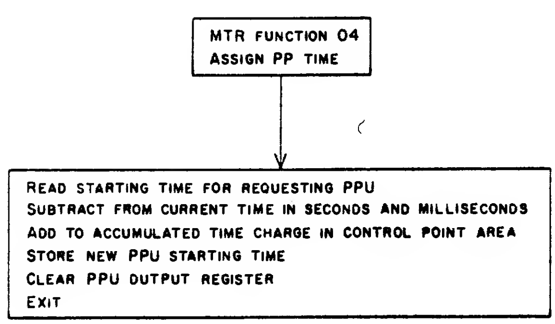



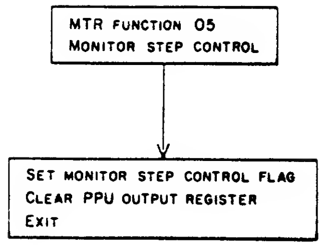

MTR SUBROUTINE
PROCESS PPU MESSAGE
IS MONITOR IN STEP MODE ?
YES
SET WAIT STEP FLAG AT CENTRAL ADDRESS 0014
NO
RJ ADVANCE CLOCK
NO
HAS OPERATOR STEPPED MONITOR ?
YES
READ FUNCTION FROM REQUESTING
PPU OUTPUT REGISTER
RJ TO CORRESPONDING MTR SUBROUTINE
CLEAR STEP FLAG
RJ ADVANCE CLOCK
EXIT

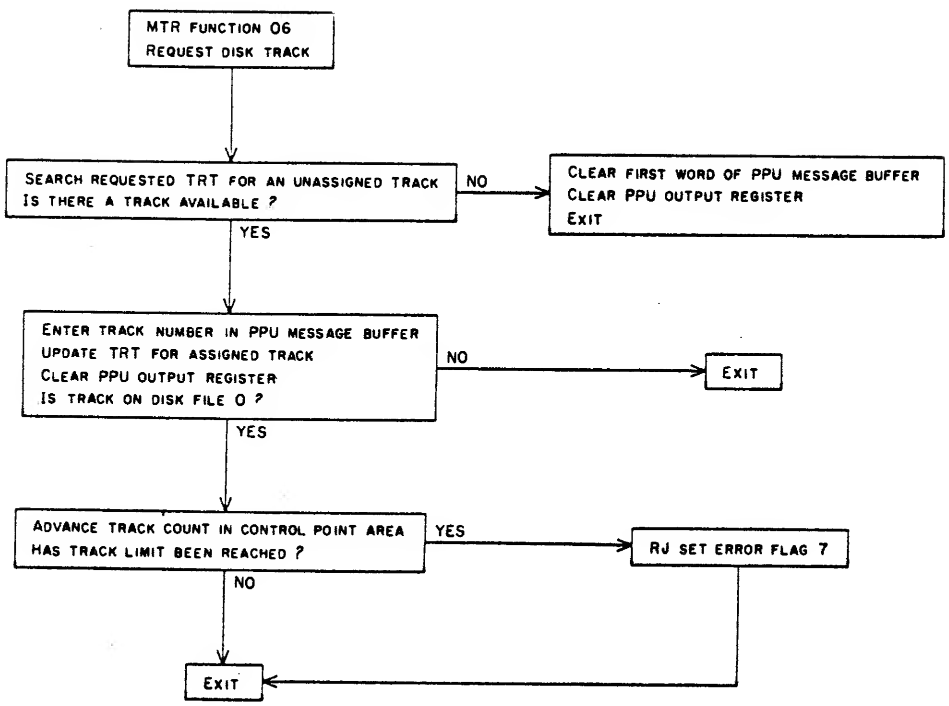

MTR FUNCTION 01
PROCESS DAYFILE MESSAGE
IS DUMP FLAG SET ?
YES
EXIT
NO
COPY MESSAGE FROM PPU MESSAGE BUFFER TO CONTROL POINT AREA
ENTER TIME IN DAYFILE BUFFER
ENTER JOB NAME IN DAYFILE BUFFER
COPY MESSAGE FROM PPU MESSAGE BUFFER TO DAYFILE BUFFER
NO
CLEAR PPU OUTPUT REGISTER
DOES DAYFILE BUFFER CONTAIN A FULL DISK SECTOR OF DATA ?
YES
SET PHASE ONE DUMP FLAG
EXIT

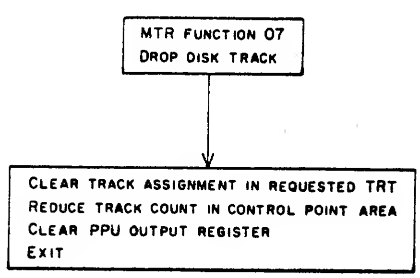

MTR FUNCTION 02
REQUEST CHANNEL
READ CHANNEL STATUS TABLE
IS REQUESTED CHANNEL BUSY ?
YES
EXIT
NO
ASSIGN CHANNEL TO REQUESTING PPU
UPDATE CHANNEL STATUS TABLE
CLEAR PPU OUTPUT REGISTER

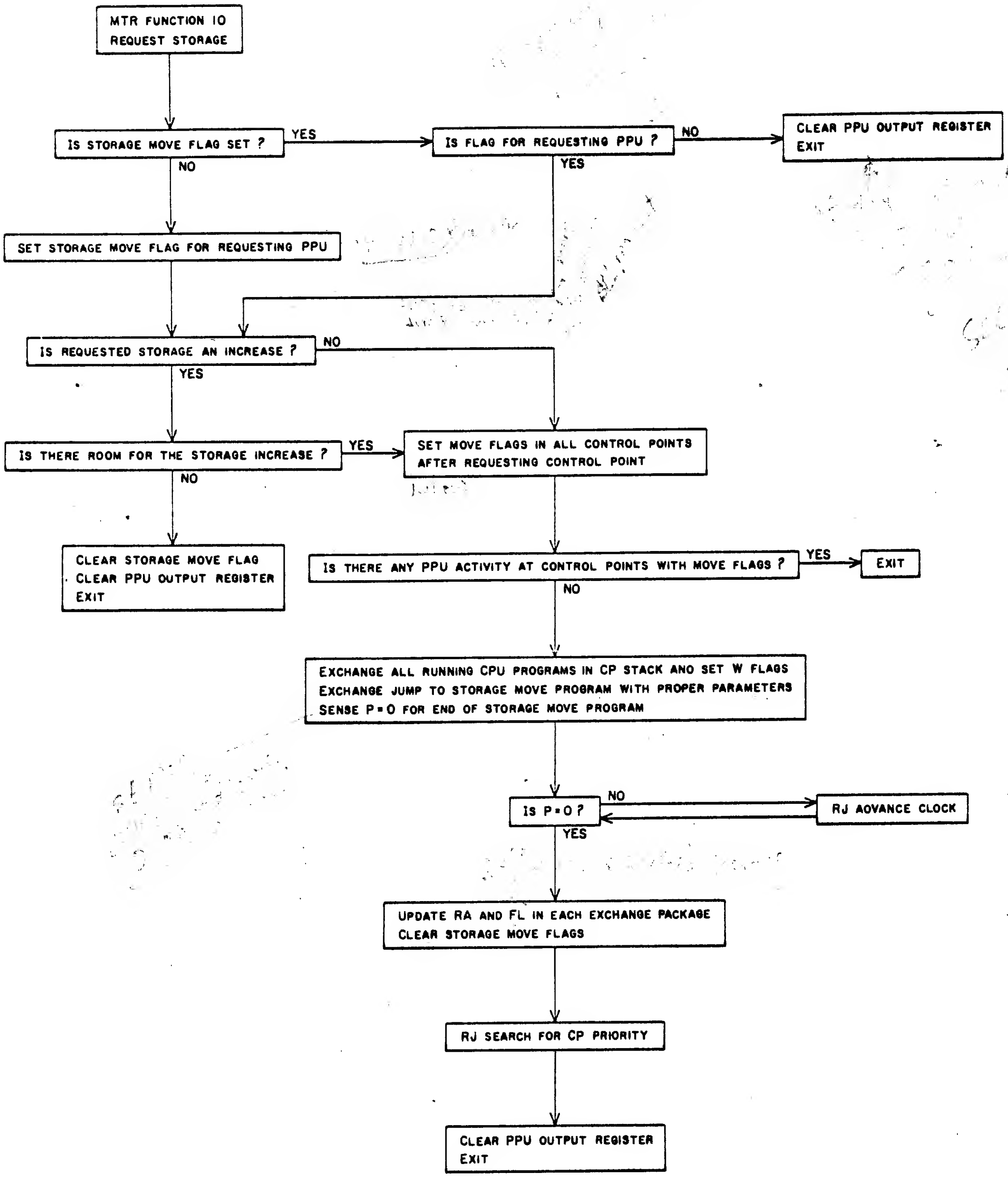

MTR FUNCTION 03
DROP CHANNEL
READ CHANNEL STATUS TABLE
CLEAR REQUESTED CHANNEL ASSIGNMENT
UPDATE CHANNEL STATUS TABLE
EXIT

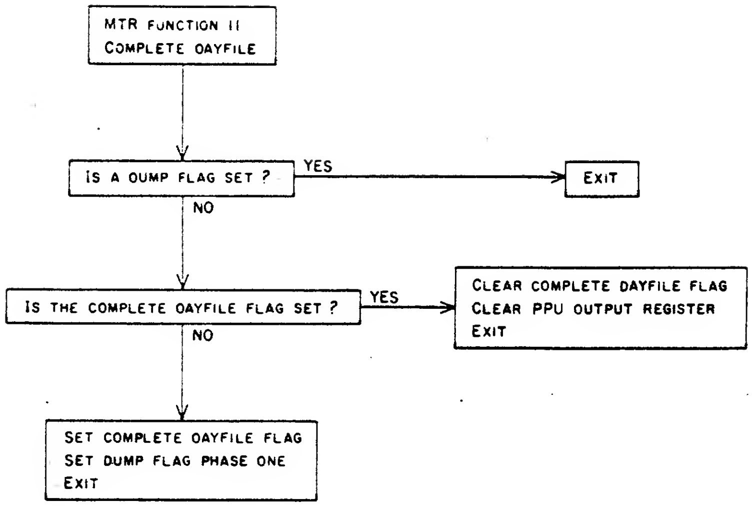

MTR FUNCTION 04
ASSIGN PP TIME
READ STARTING TIME FOR REQUESTING PPU
SUBTRACT FROM CURRENT TIME IN SECONDS AND MILLISECONDS
ADD TO ACCUMULATED TIME CHARGE IN CONTROL POINT AREA
STORE NEW PPU STARTING TIME
CLEAR PPU OUTPUT REGISTER
EXIT

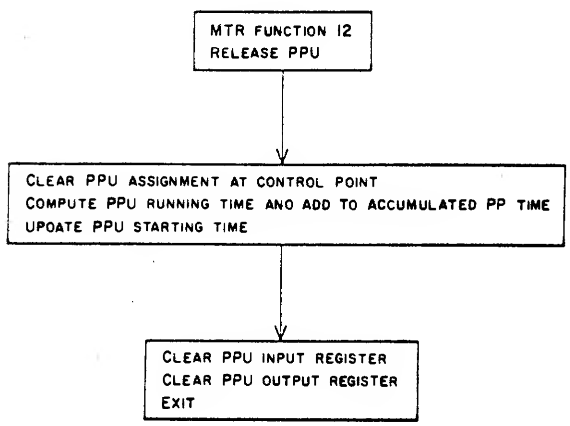

MTR FUNCTION 05
MONITOR STEP CONTROL
SET MONITOR STEP CONTROL FLAG
CLEAR PPU OUTPUT REGISTER
EXIT

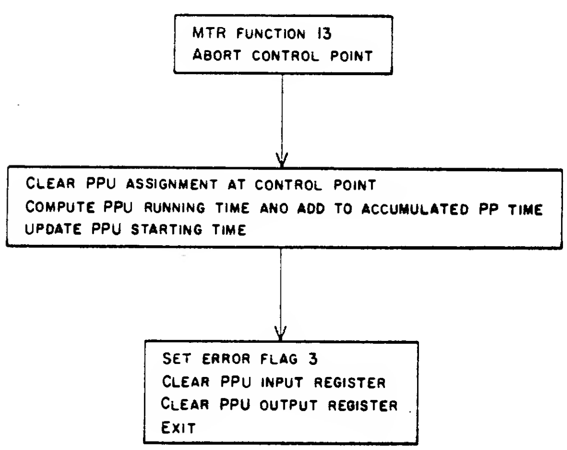

MTR FUNCTION 06
REQUEST DISK TRACK
SEARCH REQUESTED TRT FOR AN UNASSIGNED TRACK
IS THERE A TRACK AVAILABLE ?
NO
CLEAR FIRST WORD OF PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
YES
ENTER TRACK NUMBER IN PPU MESSAGE BUFFER
UPDATE TRT FOR ASSIGNED TRACK
CLEAR PPU OUTPUT REGISTER
IS TRACK ON DISK FILE 0 ?
NO
EXIT
YES
ADVANCE TRACK COUNT IN CONTROL POINT AREA
HAS TRACK LIMIT BEEN REACHED ?
YES
RJ SET ERROR FLAG 7
NO
EXIT

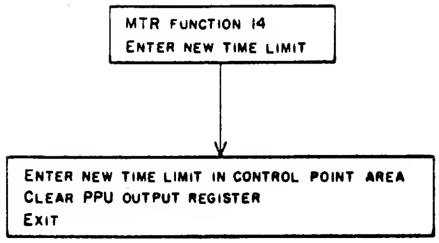

MTR FUNCTION 07
DROP DISK TRACK
CLEAR TRACK ASSIGNMENT IN REQUESTED TRT
REDUCE TRACK COUNT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

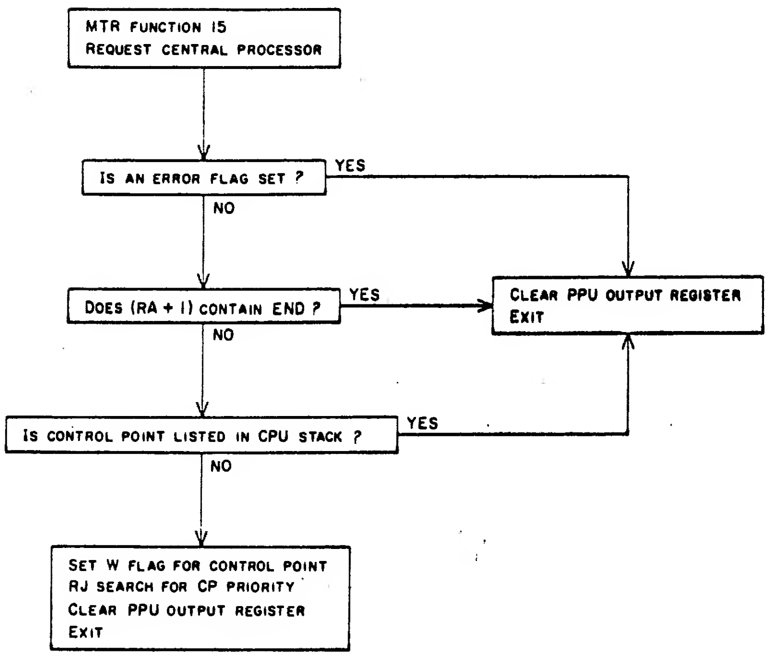

MTR FUNCTION 10
REQUEST STORAGE
IS STORAGE MOVE FLAG SET ?
YES
NO
IS FLAG FOR REQUESTING PPU ?
NO
YES
CLEAR PPU OUTPUT REGISTER
EXIT
SET STORAGE MOVE FLAG FOR REQUESTING PPU
IS REQUESTED STORAGE AN INCREASE ?
NO
YES
IS THERE ROOM FOR THE STORAGE INCREASE ?
YES
NO
SET MOVE FLAGS IN ALL CONTROL POINTS
AFTER REQUESTING CONTROL POINT
CLEAR STORAGE MOVE FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
IS THERE ANY PPU ACTIVITY AT CONTROL POINTS WITH MOVE FLAGS ?
YES
EXIT
NO
EXCHANGE ALL RUNNING CPU PROGRAMS IN CP STACK AND SET W FLAGS
EXCHANGE JUMP TO STORAGE MOVE PROGRAM WITH PROPER PARAMETERS
SENSE P = 0 FOR END OF STORAGE MOVE PROGRAM
IS P = 0 ?
NO
RJ ADVANCE CLOCK
YES
UPDATE RA AND FL IN EACH EXCHANGE PACKAGE
CLEAR STORAGE MOVE FLAGS
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

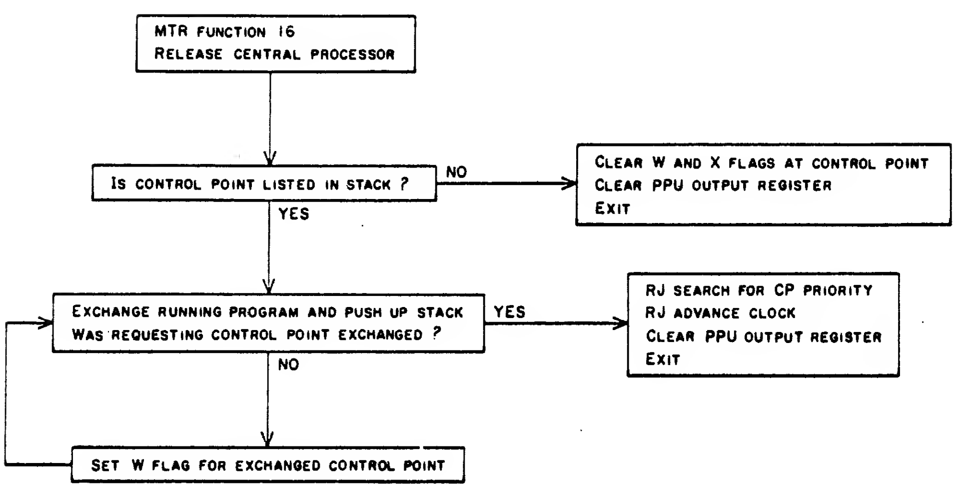

MTR FUNCTION 11
COMPLETE OAYFILE
IS A OUMP FLAG SET ?
YES
EXIT
NO
IS THE COMPLETE OAYFILE FLAG SET ?
YES
CLEAR COMPLETE DAYFILE FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SET COMPLETE OAYFILE FLAG
SET DUMP FLAG PHASE ONE
EXIT

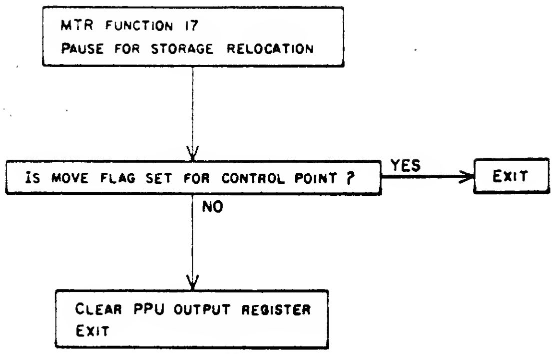

MTR FUNCTION 12
RELEASE PPU
CLEAR PPU ASSIGNMENT AT CONTROL POINT
COMPUTE PPU RUNNING TIME ANO ADD TO ACCUMULATED PP TIME
UPOATE PPU STARTING TIME
CLEAR PPU INPUT REGISTER
CLEAR PPU OUTPUT REGISTER
EXIT

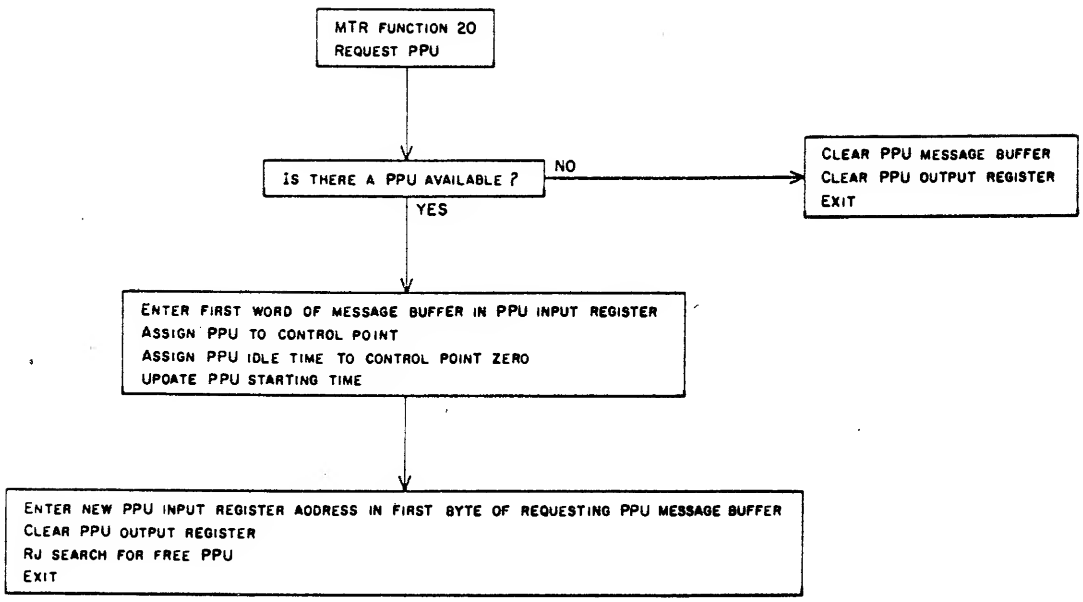

MTR FUNCTION 13
ABORT CONTROL POINT
CLEAR PPU ASSIGNMENT AT CONTROL POINT
COMPUTE PPU RUNNING TIME ANO ADD TO ACCUMULATED PP TIME
UPDATE PPU STARTING TIME
SET ERROR FLAG 3
CLEAR PPU INPUT REGISTER
CLEAR PPU OUTPUT REGISTER
EXIT

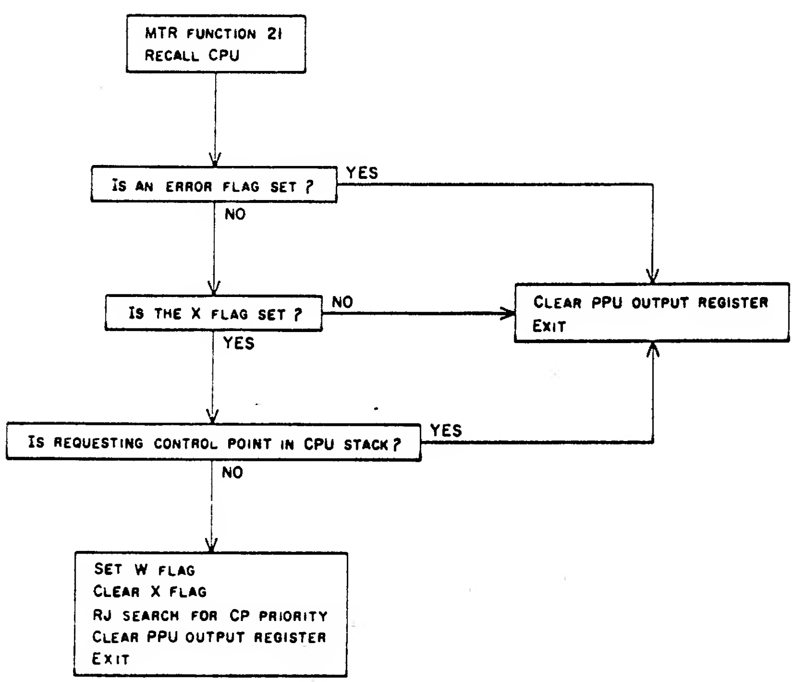

MTR FUNCTION 14
ENTER NEW TIME LIMIT
ENTER NEW TIME LIMIT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

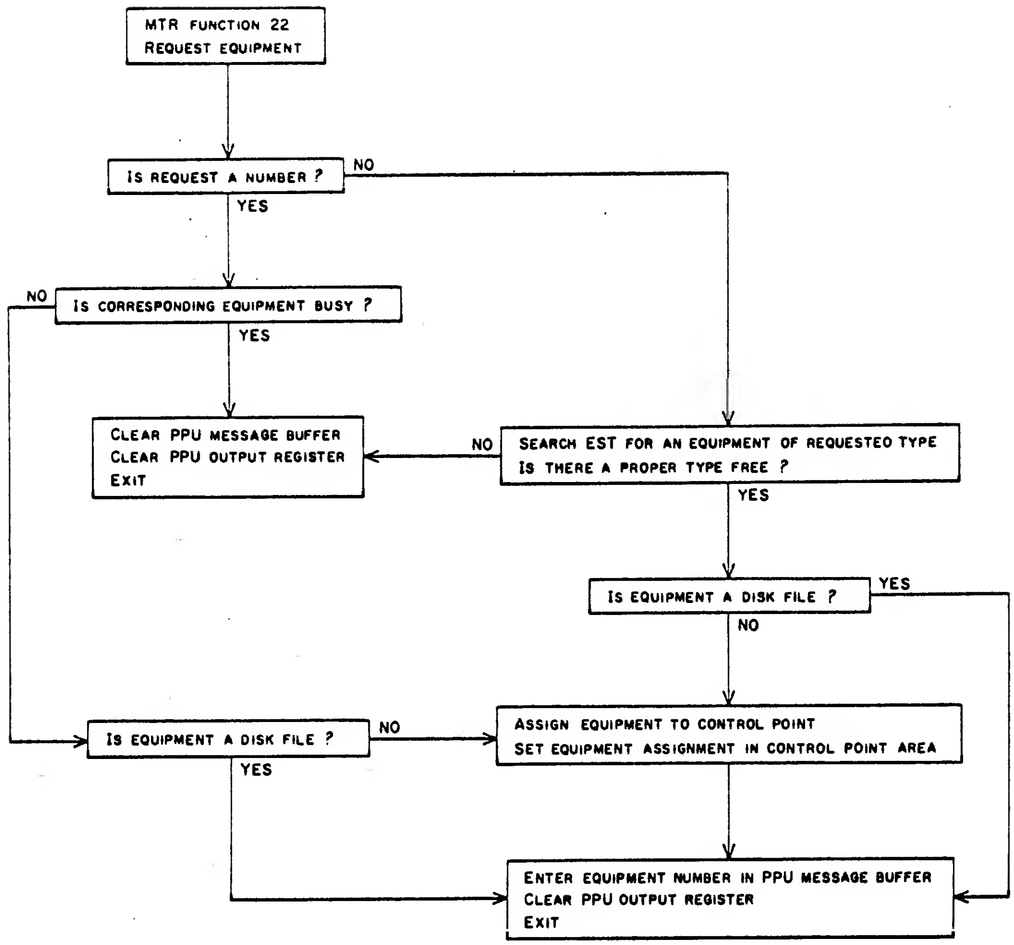

MTR FUNCTION 15
REQUEST CENTRAL PROCESSOR
IS AN ERROR FLAG SET ?
YES
NO
DOES (RA + 1) CONTAIN END ?
YES
NO
CLEAR PPU OUTPUT REGISTER
EXIT
IS CONTROL POINT LISTED IN CPU STACK ?
YES
NO
SET W FLAG FOR CONTROL POINT
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

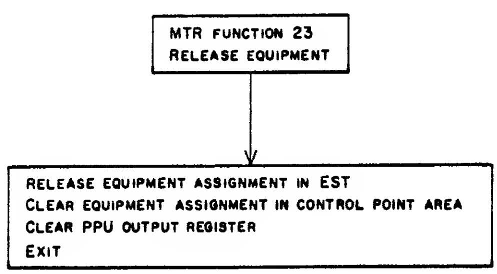

MTR FUNCTION 16
RELEASE CENTRAL PROCESSOR
IS CONTROL POINT LISTED IN STACK ?
NO
CLEAR W AND X FLAGS AT CONTROL POINT
CLEAR PPU OUTPUT REGISTER
EXIT
YES
EXCHANGE RUNNING PROGRAM AND PUSH UP STACK
WAS REQUESTING CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
RJ ADVANCE CLOCK
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SET W FLAG FOR EXCHANGED CONTROL POINT

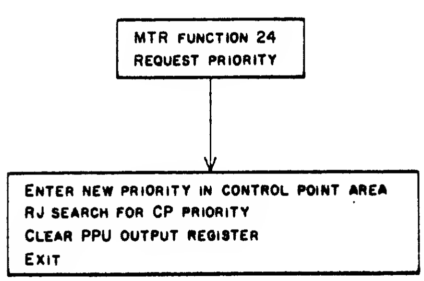

MTR FUNCTION 17
PAUSE FOR STORAGE RELOCATION
IS MOVE FLAG SET FOR CONTROL POINT ?
YES
EXIT
NO
CLEAR PPU OUTPUT REGISTER
EXIT

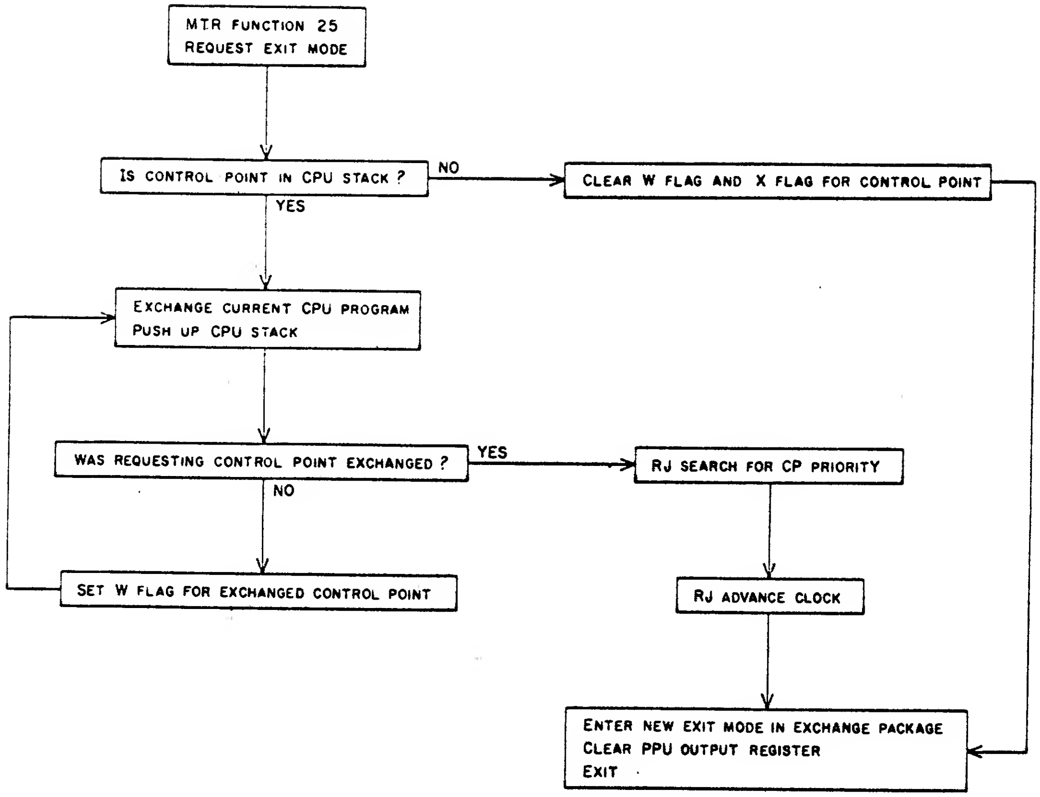

MTR FUNCTION 20
REQUEST PPU
IS THERE A PPU AVAILABLE ?
NO
CLEAR PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
YES
ENTER FIRST WORD OF MESSAGE BUFFER IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
ENTER NEW PPU INPUT REGISTER ADDRESS IN FIRST BYTE OF REQUESTING PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
RJ SEARCH FOR FREE PPU
EXIT

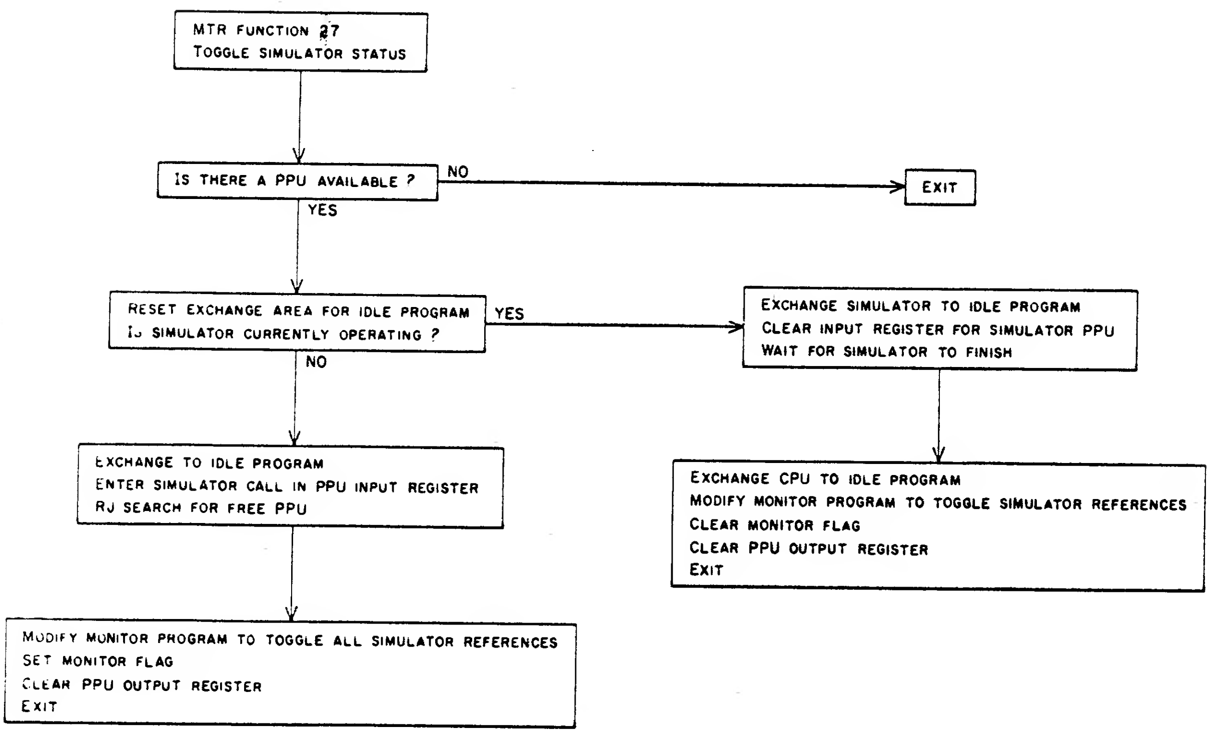

MTR FUNCTION 21
RECALL CPU
IS AN ERROR FLAG SET ?
YES
NO
IS THE X FLAG SET ?
NO
CLEAR PPU OUTPUT REGISTER
EXIT
YES
IS REQUESTING CONTROL POINT IN CPU STACK ?
YES
NO
SET W FLAG
CLEAR X FLAG
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

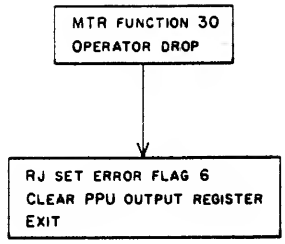

MTR FUNCTION 22
REQUEST EQUIPMENT
IS REQUEST A NUMBER ?
NO
YES
IS CORRESPONDING EQUIPMENT BUSY ?
NO
YES
CLEAR PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SEARCH EST FOR AN EQUIPMENT OF REQUESTED TYPE
IS THERE A PROPER TYPE FREE ?
YES
IS EQUIPMENT A DISK FILE ?
YES
NO
IS EQUIPMENT A DISK FILE ?
NO
YES
ASSIGN EQUIPMENT TO CONTROL POINT
SET EQUIPMENT ASSIGNMENT IN CONTROL POINT AREA
ENTER EQUIPMENT NUMBER IN PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT

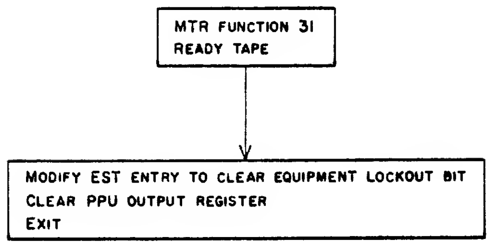

MTR FUNCTION 23
RELEASE EQUIPMENT
RELEASE EQUIPMENT ASSIGNMENT IN EST
CLEAR EQUIPMENT ASSIGNMENT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

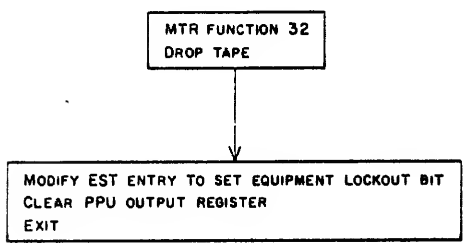

MTR FUNCTION 24
REQUEST PRIORITY
ENTER NEW PRIORITY IN CONTROL POINT AREA
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

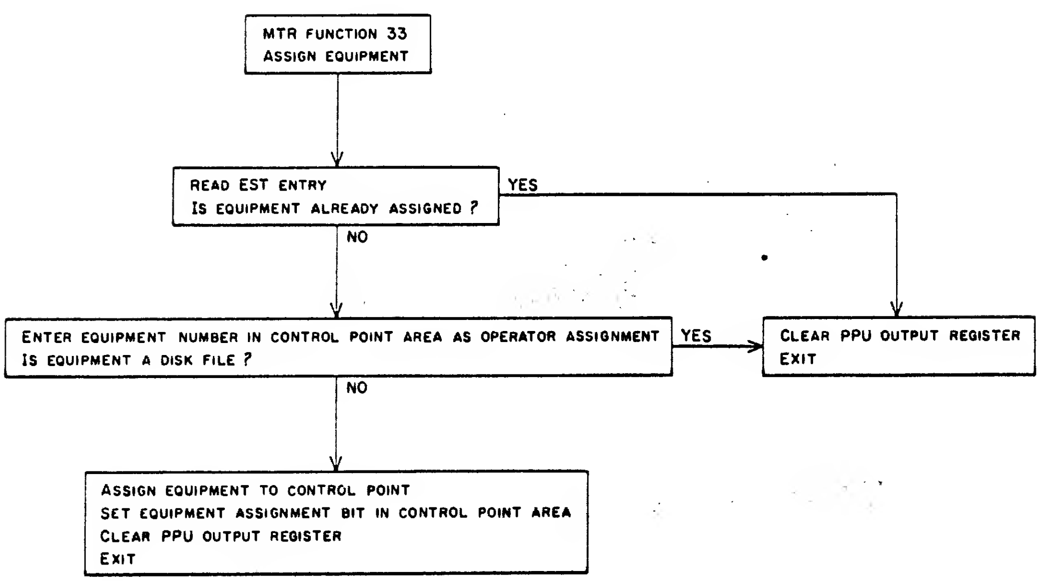

MTR FUNCTION 25
REQUEST EXIT MODE
IS CONTROL POINT IN CPU STACK ?
NO
CLEAR W FLAG AND X FLAG FOR CONTROL POINT
YES
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
WAS REQUESTING CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
NO
SET W FLAG FOR EXCHANGED CONTROL POINT
RJ ADVANCE CLOCK
ENTER NEW EXIT MODE IN EXCHANGE PACKAGE
CLEAR PPU OUTPUT REGISTER
EXIT

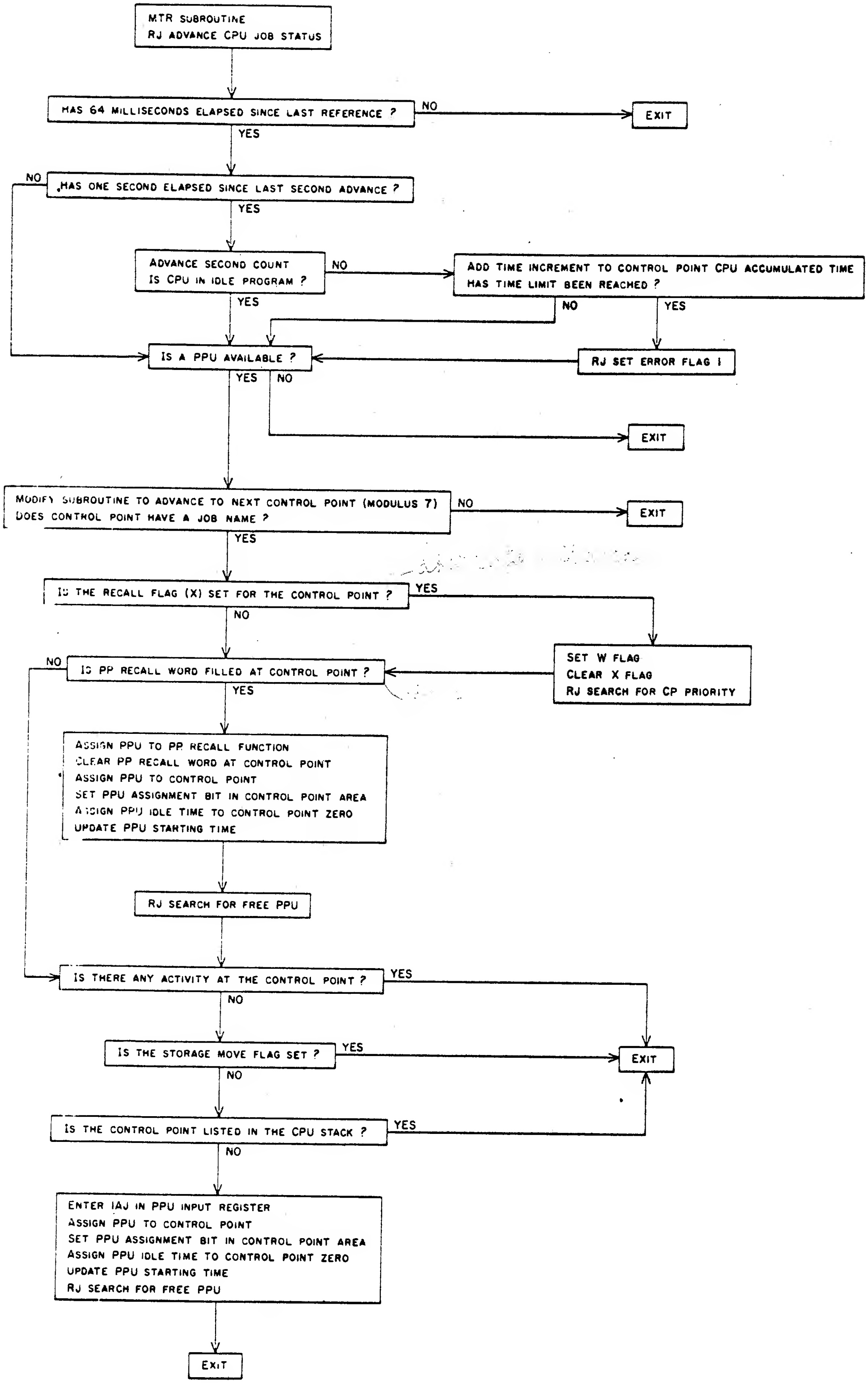

MTR FUNCTION 27
TOGGLE SIMULATOR STATUS
IS THERE A PPU AVAILABLE ?
NO
EXIT
YES
RESET EXCHANGE AREA FOR IDLE PROGRAM
IS SIMULATOR CURRENTLY OPERATING ?
YES
EXCHANGE SIMULATOR TO IDLE PROGRAM
CLEAR INPUT REGISTER FOR SIMULATOR PPU
WAIT FOR SIMULATOR TO FINISH
NO
EXCHANGE TO IDLE PROGRAM
ENTER SIMULATOR CALL IN PPU INPUT REGISTER
RJ SEARCH FOR FREE PPU
EXCHANGE CPU TO IDLE PROGRAM
MODIFY MONITOR PROGRAM TO TOGGLE SIMULATOR REFERENCES
CLEAR MONITOR FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
MODIFY MONITOR PROGRAM TO TOGGLE ALL SIMULATOR REFERENCES
SET MONITOR FLAG
CLEAR PPU OUTPUT REGISTER
EXIT

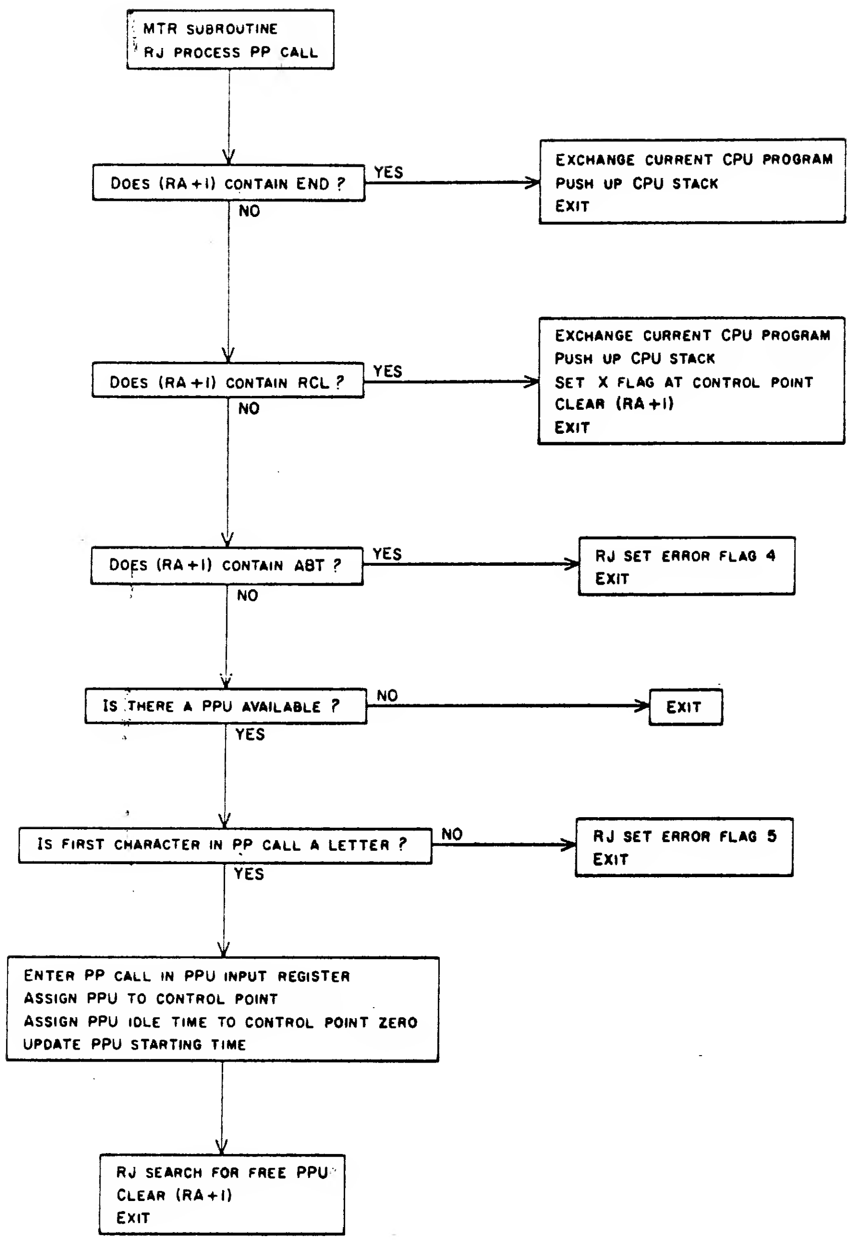

MTR FUNCTION 30
OPERATOR DROP
RJ SET ERROR FLAG 6
CLEAR PPU OUTPUT REGISTER
EXIT


MTR FUNCTION 31
READY TAPE
MODIFY EST ENTRY TO CLEAR EQUIPMENT LOCKOUT BIT
CLEAR PPU OUTPUT REGISTER
EXIT

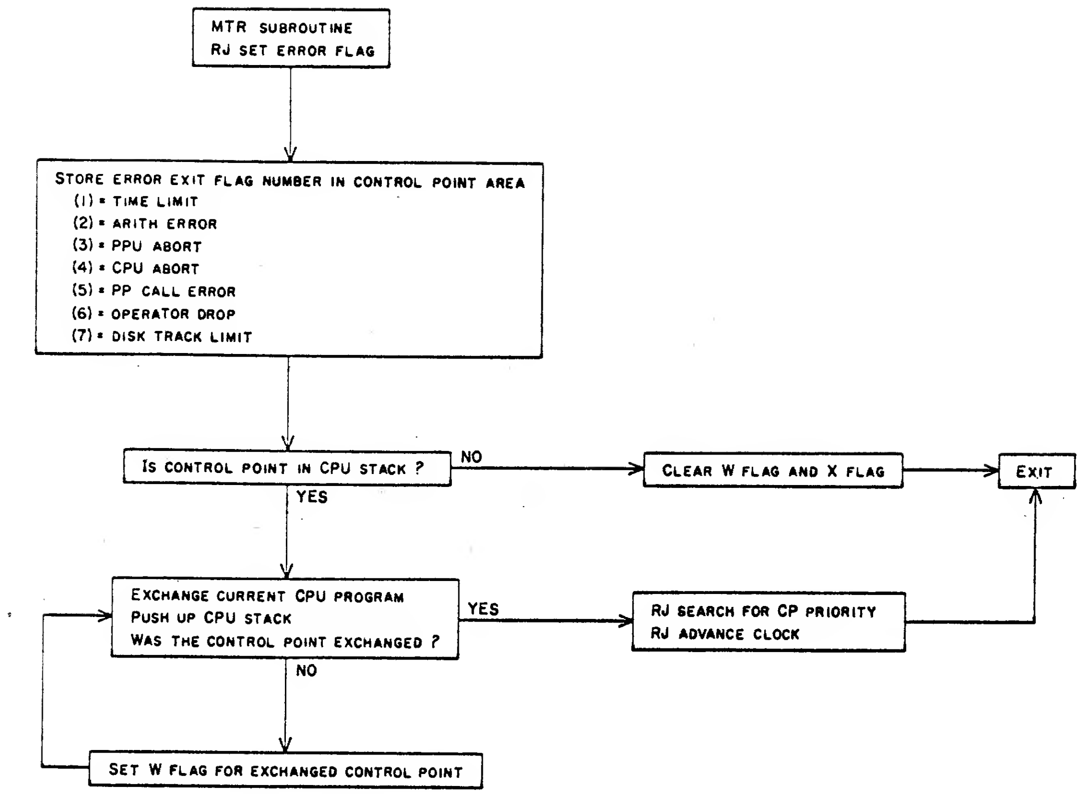

MTR FUNCTION 32
DROP TAPE
MODIFY EST ENTRY TO SET EQUIPMENT LOCKOUT BIT
CLEAR PPU OUTPUT REGISTER
EXIT

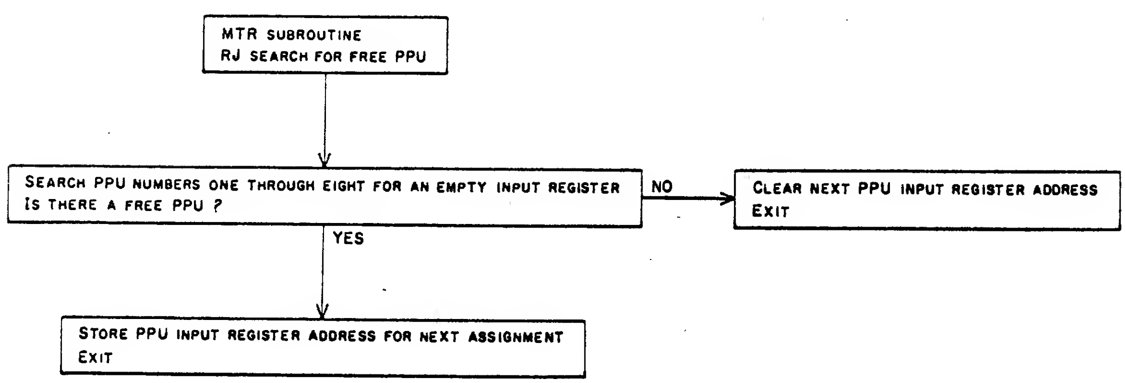

MTR FUNCTION 33
ASSIGN EQUIPMENT
READ EST ENTRY
IS EQUIPMENT ALREADY ASSIGNED ?
YES
NO
ENTER EQUIPMENT NUMBER IN CONTROL POINT AREA AS OPERATOR ASSIGNMENT
IS EQUIPMENT A DISK FILE ?
YES
CLEAR PPU OUTPUT REGISTER
EXIT
NO
ASSIGN EQUIPMENT TO CONTROL POINT
SET EQUIPMENT ASSIGNMENT BIT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

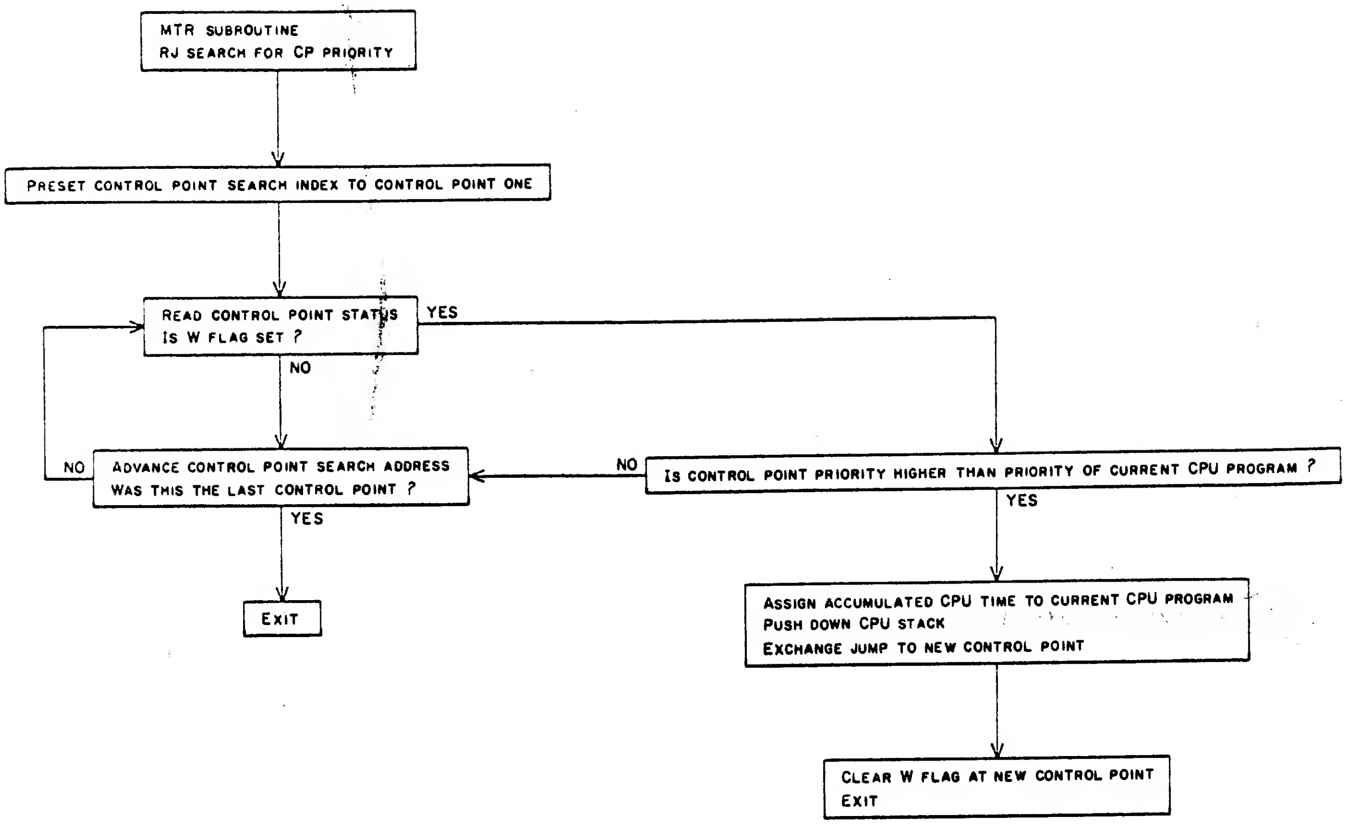

MTR SUBROUTINE
RJ ADVANCE CPU JOB STATUS
HAS 64 MILLISECONDS ELAPSED SINCE LAST REFERENCE ?
NO
EXIT
YES
HAS ONE SECOND ELAPSED SINCE LAST SECOND ADVANCE ?
NO
YES
ADVANCE SECOND COUNT
IS CPU IN IDLE PROGRAM ?
NO
ADD TIME INCREMENT TO CONTROL POINT CPU ACCUMULATED TIME
HAS TIME LIMIT BEEN REACHED ?
NO
YES
YES
IS A PPU AVAILABLE ?
RJ SET ERROR FLAG I
YES
NO
EXIT
MODIFY SUBROUTINE TO ADVANCE TO NEXT CONTROL POINT (MODULUS 7)
DOES CONTROL POINT HAVE A JOB NAME ?
NO
EXIT
YES
IS THE RECALL FLAG (X) SET FOR THE CONTROL POINT ?
YES
NO
SET W FLAG
CLEAR X FLAG
RJ SEARCH FOR CP PRIORITY
IS PP RECALL WORD FILLED AT CONTROL POINT ?
NO
YES
ASSIGN PPU TO PP RECALL FUNCTION
CLEAR PP RECALL WORD AT CONTROL POINT
ASSIGN PPU TO CONTROL POINT
SET PPU ASSIGNMENT BIT IN CONTROL POINT AREA
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
IS THERE ANY ACTIVITY AT THE CONTROL POINT ?
YES
NO
IS THE STORAGE MOVE FLAG SET ?
YES
EXIT
NO
IS THE CONTROL POINT LISTED IN THE CPU STACK ?
YES
NO
ENTER IAJ IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
SET PPU ASSIGNMENT BIT IN CONTROL POINT AREA
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
EXIT

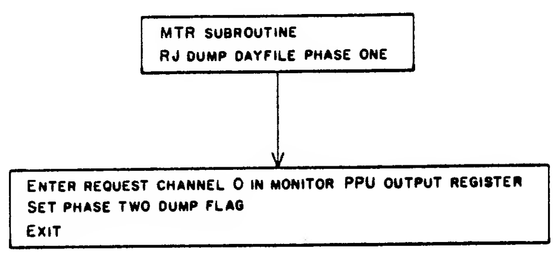


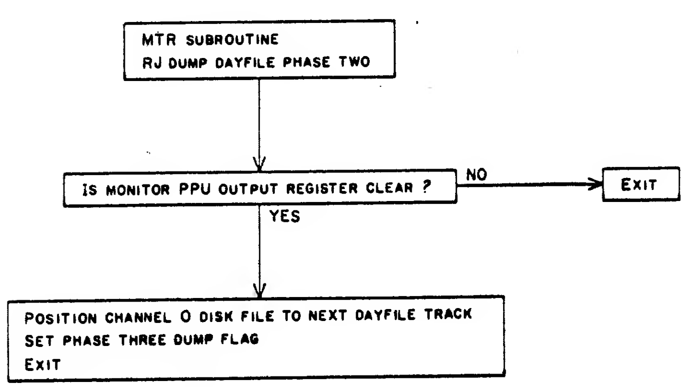
MTR SUBROUTINE
RJ PROCESS PP CALL
DOES (RA+1) CONTAIN END ?
YES
NO
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
EXIT
DOES (RA+1) CONTAIN RCL ?
YES
NO
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
SET X FLAG AT CONTROL POINT
CLEAR (RA+1)
EXIT
DOES (RA+1) CONTAIN ABT ?
YES
NO
RJ SET ERROR FLAG 4
EXIT
IS THERE A PPU AVAILABLE ?
NO
YES
EXIT
IS FIRST CHARACTER IN PP CALL A LETTER ?
NO
YES
RJ SET ERROR FLAG 5
EXIT
ENTER PP CALL IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
CLEAR (RA+1)
EXIT

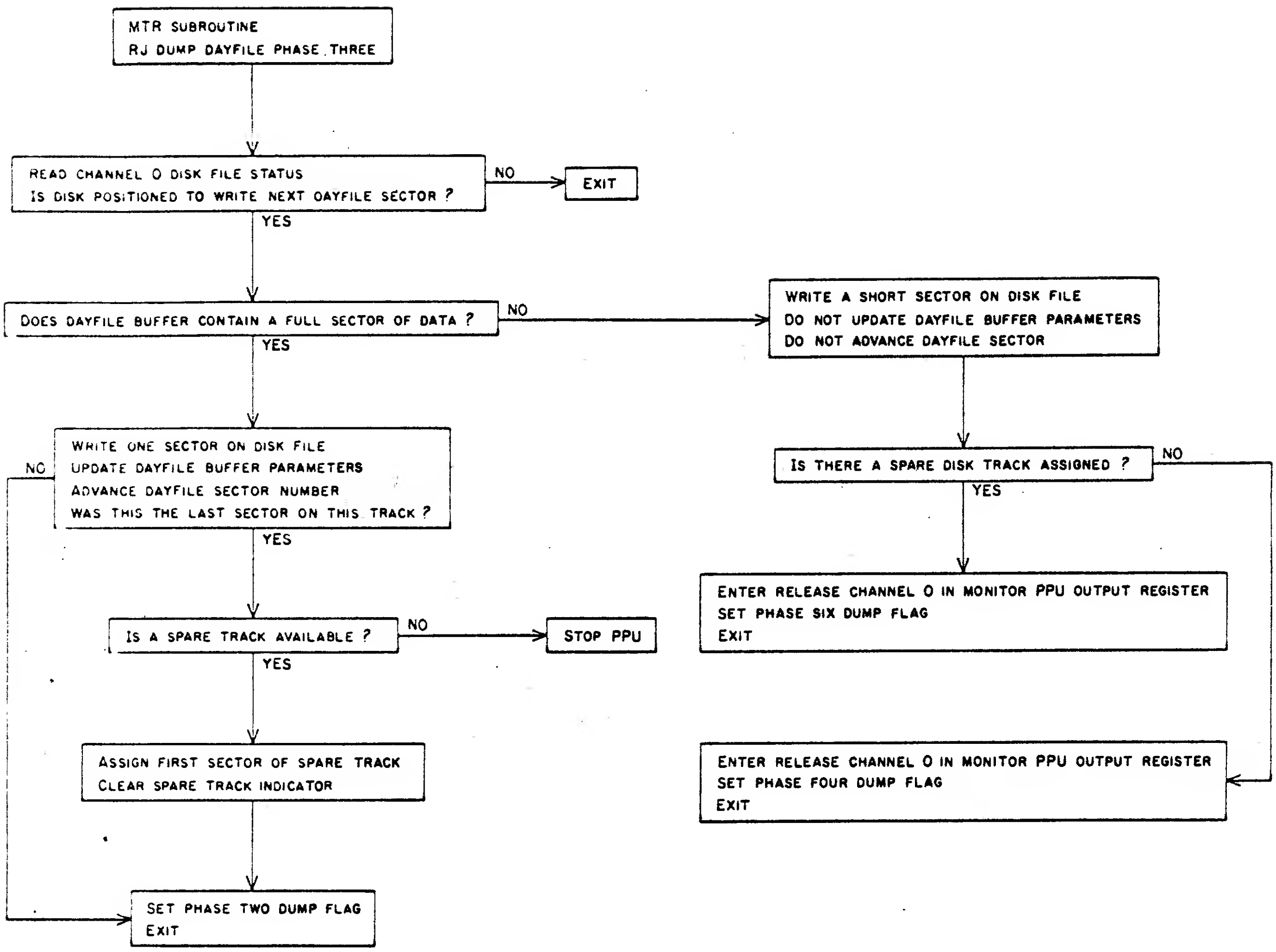

MTR SUBROUTINE
RJ SET ERROR FLAG
STORE ERROR EXIT FLAG NUMBER IN CONTROL POINT AREA
(1) = TIME LIMIT
(2) = ARITH ERROR
(3) = PPU ABORT
(4) = CPU ABORT
(5) = PP CALL ERROR
(6) = OPERATOR DROP
(7) = DISK TRACK LIMIT
IS CONTROL POINT IN CPU STACK ?
NO
CLEAR W FLAG AND X FLAG
EXIT
YES
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
WAS THE CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
RJ ADVANCE CLOCK
NO
SET W FLAG FOR EXCHANGED CONTROL POINT

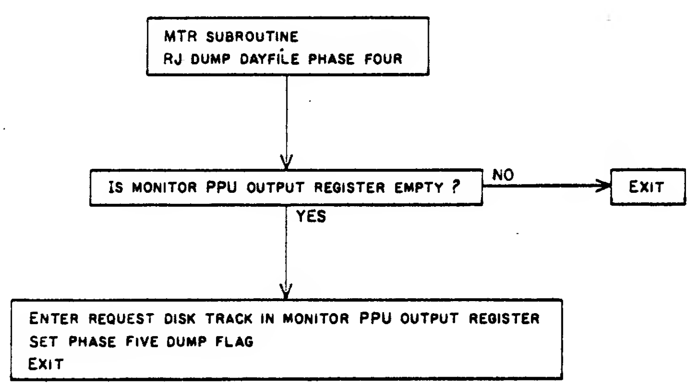

MTR SUBROUTINE
RJ SEARCH FOR FREE PPU
SEARCH PPU NUMBERS ONE THROUGH EIGHT FOR AN EMPTY INPUT REGISTER
IS THERE A FREE PPU ?
NO
CLEAR NEXT PPU INPUT REGISTER ADDRESS
EXIT
YES
STORE PPU INPUT REGISTER ADDRESS FOR NEXT ASSIGNMENT
EXIT

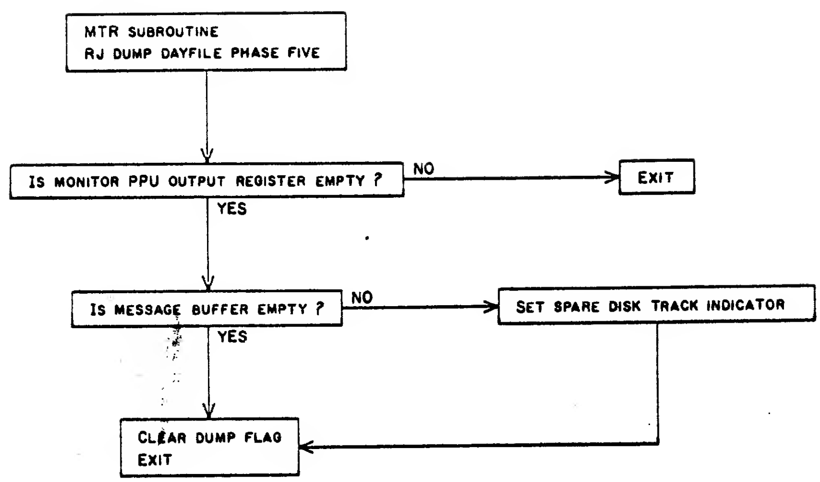

MTR SUBROUTINE
RJ SEARCH FOR CP PRIORITY
PRESET CONTROL POINT SEARCH INDEX TO CONTROL POINT ONE
READ CONTROL POINT STATUS
IS W FLAG SET ?
YES
NO
ADVANCE CONTROL POINT SEARCH ADDRESS
WAS THIS THE LAST CONTROL POINT ?
NO
YES
EXIT
IS CONTROL POINT PRIORITY HIGHER THAN PRIORITY OF CURRENT CPU PROGRAM ?
NO
YES
ASSIGN ACCUMULATED CPU TIME TO CURRENT CPU PROGRAM
PUSH DOWN CPU STACK
EXCHANGE JUMP TO NEW CONTROL POINT
CLEAR W FLAG AT NEW CONTROL POINT
EXIT

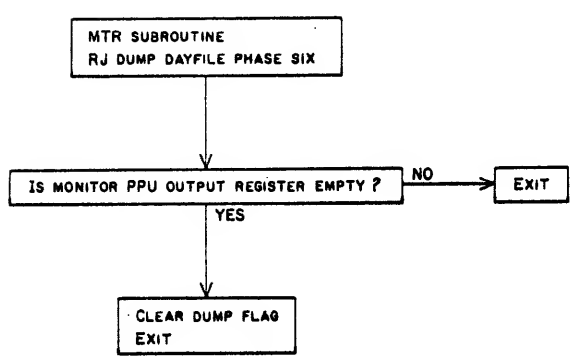

MTR SUBROUTINE
RJ DUMP DAYFILE PHASE ONE
ENTER REQUEST CHANNEL 0 IN MONITOR PPU OUTPUT REGISTER
SET PHASE TWO DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE TWO
IS MONITOR PPU OUTPUT REGISTER CLEAR ?
NO
EXIT
YES
POSITION CHANNEL 0 DISK FILE TO NEXT DAYFILE TRACK
SET PHASE THREE DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE THREE
READ CHANNEL O DISK FILE STATUS
IS DISK POSITIONED TO WRITE NEXT DAYFILE SECTOR ?
NO
EXIT
YES
DOES DAYFILE BUFFER CONTAIN A FULL SECTOR OF DATA ?
NO
WRITE A SHORT SECTOR ON DISK FILE
DO NOT UPDATE DAYFILE BUFFER PARAMETERS
DO NOT ADVANCE DAYFILE SECTOR
YES
WRITE ONE SECTOR ON DISK FILE
UPDATE DAYFILE BUFFER PARAMETERS
ADVANCE DAYFILE SECTOR NUMBER
WAS THIS THE LAST SECTOR ON THIS TRACK ?
NO
YES
IS THERE A SPARE DISK TRACK ASSIGNED ?
NO
YES
ENTER RELEASE CHANNEL O IN MONITOR PPU OUTPUT REGISTER
SET PHASE SIX DUMP FLAG
EXIT
IS A SPARE TRACK AVAILABLE ?
NO
STOP PPU
YES
ASSIGN FIRST SECTOR OF SPARE TRACK
CLEAR SPARE TRACK INDICATOR
ENTER RELEASE CHANNEL O IN MONITOR PPU OUTPUT REGISTER
SET PHASE FOUR DUMP FLAG
EXIT
SET PHASE TWO DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE FOUR
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
ENTER REQUEST DISK TRACK IN MONITOR PPU OUTPUT REGISTER
SET PHASE FIVE DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE FIVE
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
IS MESSAGE BUFFER EMPTY ?
NO
SET SPARE DISK TRACK INDICATOR
YES
CLEAR DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE SIX
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
CLEAR DUMP FLAG
EXIT

## STORAGE MOVE PROGRAM

1. The exchange package for the storage move program is set by MTR as follows:

   P = 2022
   RA = 0
   FL = 400000
   B1 = RA + FL for requesting control point
   B2 = RA + FL for control point 7
   B3 = Increase or decrease

2. The coding for the storage move program is shown below.

| Location | Instruction | Remarks |
|---|---|---|
| 2020 | CØN 0 | |
| 2021 | CØN 0 | |
| 2022 | SB7 = 1 | |
| | EQ B1, B2, 0 | Exit if control point 7 is the requestor |
| 2023 | LT B3,B0,2027 | Jump if decrease (B3 negative) |
| | NØ | Pass |
| | NØ | Pass |
| 2024 | SA1 = B2 - B7 | "Shuttle up" loop |
| | SA2 = A1 - B7 | |
| | BX6 = X1 | |
| | BX7 = X2 | |
| 2025 | SA6 = A1 + B3 | |
| | SA7 = A2 + B3 | |
| | SB2 = B2 - 2 | |
| 2026 | NE B2,B1,2024 | |
| | JP 0 | Go to stop with P = 0 |
| 2027 | SA1 = B1 | "Shuttle down" loop |
| | SA2 = B1 + B7 | |
| | BX6 = X1 | |
| | BX7 = X2 | |
| 2030 | SA6 = A1 + B3 | |
| | SA7 = A2 + B3 | |
| | SB1 = B1 + 2 | |
| 2031 | NE B2,B1,2027 | |
| | JP 0 | Go to stop with P = 0 |

3. Timing for the "Shuttle up" loop of the storage move program is as follows:

| Location | Instruction | Issue | Begin Execution | Result Avail. | Unit Avail. |
|---|---|---|---|---|---|
| 2024 | SA1 = B2 - B7 | 0 | 0 | 3(A) 8(X) | 4 |
| | SA2 = A1 - B7 | 2 | 3 | 6(A) 11(X) | 7 |
| | BX6 = X1 | 3 | 8 | 11 | 12 |
| | BX7 = X2 | 12 | 12 | 15 | 16 |
| 2025 | SA6 = A1 + B3 | 13 | 13 | 16 | 17 |
| | SA7 = A2 + B3 | 15 | 15 | 18 | 19 |
| | SB2 = B2 - 2 | 17 | 17 | 20 | 21 |
| 2026 | NE B2,B1, 2024 | 19 | 20 | 28 | - |

Note: timing is in minor cycles - loop time approximately 2.8 microseconds for transfer of two words.

CONTROL DATA CORPORATION

Development Division - Applications

# CENTRAL MEMORY RESIDENT

Chippewa Operating System

# CENTRAL MEMORY RESIDENT

## INTRODUCTION

The Chippewa Operating System uses a portion of central memory to store various types of libraries, tables, and flags: storing these tables and libraries in central memory allows them to be readily accessed by any peripheral processor. The central memory resident is illustrated in figure 1. Generally, the resident occupies central memory locations 0 - $13777_8$. Resident elements between locations 0 and $2077_8$ are directly addressed: resident elements above location $2077_8$ are addressed via pointers contained in central memory locations 1 - $12_8$. Since the major portion of the resident is relatively addressed, the size of the resident can readily be reduced or expanded to met installation requirements.

## CM RESIDENT: LOCATIONS 0 - 57

Central memory location 0 always contains a full word of zeroes: a peripheral processor may read this location in order to clear a 5-byte area in its memory. Central memory locations 1 - $12_8$ contain pointers to various libraries, tables, and pointers.

The Channel Status Table, illustrated in figure 2, occupies locations 15, 16, and 17. Each of the 12 data channels is represented by a byte in this table. If a data channel is not in use, the corresponding byte is cleared: if a data channel is being used by a peripheral processor program, the processor number (in display code) is entered in the byte for that channel. The channel number for a particular equipment is obtained by a peripheral processor program from the Equipment Status

| Address | Contents |
|---|---|
| | RESIDENT PERIPHERAL LIBRARY (RPL) |
| 7000 | RESIDENT SUBR. LIBRARY (RSL) p. 154 |
| 5000 | DAYFILE BUFFER (DFB) p. 152 |
| 4000 | FILE NAME/FILE STATUS TABLE (FNT/FST) p. 148 |
| 3200 | TRT 4 |
| 3000 | TRT 3 |
| 2700 | TRACK RESERVATION TABLE 2 |
| 2600 | TRACK RESERVATION TABLE 1 p. 144 |
| 2500 | TRACK RESERVATION TABLE 0 |
| 2400 | PERIPHERAL LIBRARY DIRECTORY (PLD) $100_8$ p. 141 |
| 2200 | CENTRAL LIBRARY DIRECTORY (CLD) $200_8$ |
| 2100 | EQUIPMENT STATUS TABLE p. 138 |
| 2000 | CENTRAL PROCESSOR RESIDENTS |
| 0200 | CONTROL POINT AREAS 7 |
| 0060 | PP COMMUNICATION AREAS |
| 0000 | POINTERS AND FLAGS |

| Address | | | | | |
|---|---|---|---|---|---|
| 56 - 57 | CP STACK INDICATORS | | | | |
| 40 - 52 | PERIPHERAL AND CENTRAL PROCESSOR STARTING TIMES | | | | |
| 30 - 37 | TIME AND DATE | | | | |
| 27 | SIMULATOR P ADDRESS | | | | |
| 26 | SIMULATOR XJ ADDRESS | | | | |
| 25 | | | | | |
| 24 | PP IDLE TIME | | | | |
| 23 | CPU IDLE TIME | | | | |
| 22 | | | | | |
| 21 | CP0 JOB NAME: "MONITOR" | | | | |
| 20 | CONTROL POINT ZERO STATUS | | | | |
| 15 - 17 | CHANNEL STATUS TABLE | | | | |
| 14 | TRT4 | MTR STEP FLAG | | | |
| 13 | TRT3 | | | | |
| 12 | TRT2 | LAST TRACK | | $100_8$ | $62_8$ |
| 11 | TRT1 | LAST TRACK | | $100_8$ | $62_8$ |
| 10 | TRT0 | LAST TRACK | | $100_8$ | $62_8$ |
| 7 | CLD | LIMIT | | | |
| 6 | RSL | LIMIT | | | |
| 5 | EST | LIMIT | | | |
| 4 | FNT | LIMIT | | | |
| 3 | DFB | IN | OUT | LIM. | |
| 2 | PLD | LIMIT | | | |
| 1 | RPL | | | | |
| 0 | 00000000 | | | | 0 |

POINTERS

Figure 1

CENTRAL MEMORY RESIDENT (TYPICAL)

Table (EST) entry for that equipment. The program transmits its request for that channel to MTR via its resident: if the table byte corresponding to that channel is cleared, MTR enters the number of the requesting processor in that byte and notifies the requestor that the channel has been assigned. When the requesting processor completes its operation on that channel, it requests MTR to drop the channel assignment, and MTR clears the corresponding byte in the table.

The first 12 bytes in the Channel Status Table correspond to the 12 data channels: the next two bytes refer to pseudo-channels 14 and 15. These two pseudo-channels serve as an interlock to the File Name Table/File Status Table. Pseudo-channel 15 controls access to File Name Table (FNT) entries: pseudo-channel 14 controls access to File Status Table (FST) entries. Peripheral processor programs request these pseudo-channel assignments in the same manner as data channel assignments are requested. Not all accesses to the FNT/FST entries require channel reservation: the function of the interlock scheme is to prevent two (or more) processors from attempting to modify the same entry at the same time. Pseudo-channel reservations are required in the following cases:

- whenever an entry is added to the FNT/FST
- whenever a file is assigned to a control point (FNT entry modified)
- whenever the buffer status byte is initialized at the beginning of an operation (FST entry modified)

Once the appropriate pseudo-channel reservation has been acknowledged by MTR, the requesting program may proceed to perform the desired modification: upon completion, the pseudo-channel reservation should be dropped by issuing the appropriate request to MTR.

The remaining locations in this portion of the central memory resident are used by MTR for flags, indicators, and temporary storage.

PSEUDO-CHANNEL FOR FST ACCESS CONTROL

PSEUDO-CHANNEL FOR FNT ACCESS CONTROL

| | | | | | |
|---|---|---|---|---|---|
| CM LOCATION 17 | CHANNEL 12 | CHANNEL 13 | CHANNEL 14 | CHANNEL 15 | |
| CM LOCATION 16 | CHANNEL 5 | CHANNEL 6 | CHANNEL 7 | CHANNEL 10 | CHANNEL 11 |
| CM LOCATION 15 | CHANNEL 0 | CHANNEL 1 | CHANNEL 2 | CHANNEL 3 | CHANNEL 4 |
| | 12 | 12 | 12 | 12 | 12 |

- CHANNEL NOT IN USE: CORRESPONDING TABLE BYTE CONTAINS ZERO
- CHANNEL IN USE: CORRESPONDING BYTE CONTAINS USER PP NUMBER

*CHANNEL STATUS TABLE*

Figure 2

CM RESIDENT: LOCATIONS 60 - 177; PP COMMUNICATIONS AREAS

These central memory locations contain ten peripheral processor communication areas, one for each processor. The communication areas are illustrated in figure 3. There are eight words in each communication area:

word 1 . . . . . . . . . Input Register (IR)

word 2 . . . . . . . . . Output Register (OR)

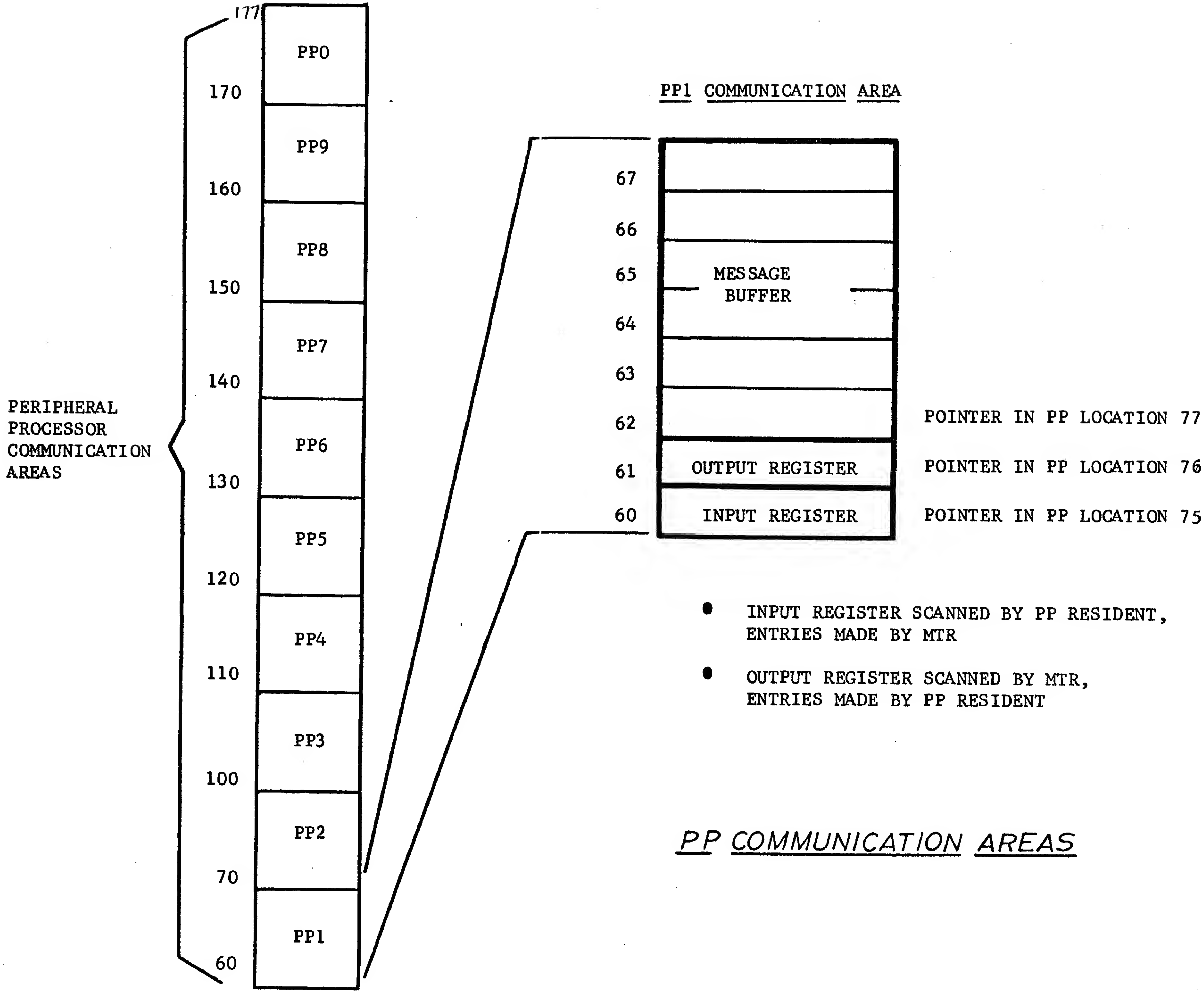
words 3 - 8 . . . . . Message Buffer (MB)

Each peripheral processor contains pointers to its Input Register, Output Register, and Message Buffer in peripheral processor memory locations 75, 76, and 77; respectively. The communication areas are used to provide a means of communication between MTR and peripheral processor programs. When a peripheral processor is idle, its resident program continuously scans its Input Register. When MTR has a task for that processor, it sets the name of the appropriate routine in the Input Register of the idle processor, which, when it recognizes the request, loads the routine and executes it. MTR regularly scans the Output Register of each peripheral processor. When a peripheral processor program requires MTR assistance (such as, for example, reserving a data channel), it places a code in its Output Register. MTR detects the request during its scan of the output registers and processes it. When the request has been processed, MTR clears the requesting processor's Output Register: this informs the requesting processor that the request has been processed.

The six-word Message Buffer is used to pass parameters and messages between MTR and the peripheral processor resident programs.

CM RESIDENT LOCATIONS 200 - 1777; CONTROL POINT AREAS

Central memory locations 200 - $1777_8$ contain seven control point areas, one for each control point. Each control point area occupies $200_8$ locations. The first $20_8$ words of a control point area contain the exchange jump package for the central processor program which may be associated with this control point. The next $10_8$ words contain various flags, status indicators, counters,




- INPUT REGISTER SCANNED BY PP RESIDENT, ENTRIES MADE BY MTR
- OUTPUT REGISTER SCANNED BY MTR, ENTRIES MADE BY PP RESIDENT

PP COMMUNICATION AREAS

Figure 3

etc., which pertain to this particular job. Another $10_8$ words are used to store the most recent console or dayfile message. The remaining $140_8$ locations are used to hold the control statements for the job assigned to this control point.

CM RESIDENT: LOCATIONS 2000 - 2077; CP RESIDENT

There are two resident central processor programs: a storage move program of some 24 instructions, and a two-instruction idle program. These two programs, together with their exchange jump packages, occupy locations 2000 - $2077_8$ of the central memory resident.

CM RESIDENT: THE EQUIPMENT STATUS TABLE

The Equipment Status Table (EST) contains a one-word entry for each peripheral device. The table occupies $64_{10}$ locations: its base address is provided to the system by the EST pointer in central memory location 5. The format of the EST entry is shown in figure 4. The first (leftmost) byte contains zero if the equipment is not assigned: if the equipment is assigned to a job at a given control point, this byte contains the control point address (in hundreds). The second byte contains the channel number for this equipment, while the third byte contains the controller and unit number in the form required by the function codes for this equipment. Byte 4 contains the equipment type in display code: each type of equipment is assigned a two-letter code, as shown below.

| | |
|---|---|
| DA . . . . . Disk 0 | CR . . . . . Card Reader |
| DB . . . . . Disk 1 | CP . . . . . Card Punch |
| DC . . . . . Disk 2 | MT . . . . . Magnetic Tape (607) |
| DS . . . . . Display | WT . . . . . Magnetic Tape (626) |

LP . . . . . . Printer

The $2^{11}$ bit in byte 4 of the EST entry is used as an operator controlled interlock for equipment availability. If this bit is zero, the equipment defined by

|ch|M|CD|cd| if channels assigned to one I/O unit (e.g. disk)

| 12 | 12 | 12 | 1 | 11 | 12 |
|---|---|---|---|---|---|
| CP ADDRESS | CHANNEL NUMBER | CONTROLLER & UNIT NO. | I | EQUIP. CODE | |

CONTROL POINT ADDRESS ~~IN HUNDREDS~~

CHANNEL NUMBER IN OCTAL (RIGHT JUSTIFIED)

CONTROLLER AND UNIT NUMBER IN FORMAT FOR INSERTION IN FUNCTION CODE

INTERLOCK BIT CONTROLLED BY OPERATOR VIA DSD:
"1" = EQUIP. NOT AVAILABLE
"0" = EQUIP. AVAILABLE

EQUIPMENT TYPE IN DISPLAY CODE:
DA = DISK 0, ETC.
DB = DISK 6638

RESERVED FOR USE WITH THE 6681

J E UU
6681 # controller # unit #

- ONE EST ENTRY FOR EACH PERIPHERAL DEVICE
- EQUIPMENT NUMBER DEFINES LOCATION OF ENTRY IN TABLE

EST ENTRY

Figure 4

by this entry is available for assignment. If this bit is one, the equipment is not available. This bit is set or cleared by use of the DSD keyboard entries OFF and ON.* Byte 5 of the EST entry is reserved for use with equipments connected via the 6681 data channel converter.

As an example of an EST entry, suppose 607-B unit 3 on the first controller on channel six is available and not assigned to a control point: the EST entry would appear as follows:

| | | | | |
|---|---|---|---|---|
| 0000 | 0006 | 2003 | 5524* | 0000 |

*Display Code for MT

Within the system, equipments are identified by an equipment number. The equipment number for a given device is the relative address in the Equipment Status Table of the entry for that device.

To illustrate the use of the Equipment Status Table, consider the processing of the control statement

ASSIGN MT, INFILE

When the statement translator (2TS overlay) processes this statement, it requests MTR to assign an equipment of this type. MTR searches the EST until an entry with the equipment type (byte 4) equal to MT is found. If this equipment is not assigned (byte 1 = 0), MTR enters the control point address in byte 1 and returns the relative location of this entry in the Equipment Status Table to the statement translator. This relative location is the equipment number: the statement translator inserts this number in byte 1 of the FST entry for this file. The routines called to process this file at some later time will use this equipment number to obtain the EST entry from the table, and from the EST entry will obtain in turn the channel number and the controller and unit number.

* The interlock bit is set to "1" at load time for equipment types MT and WT.

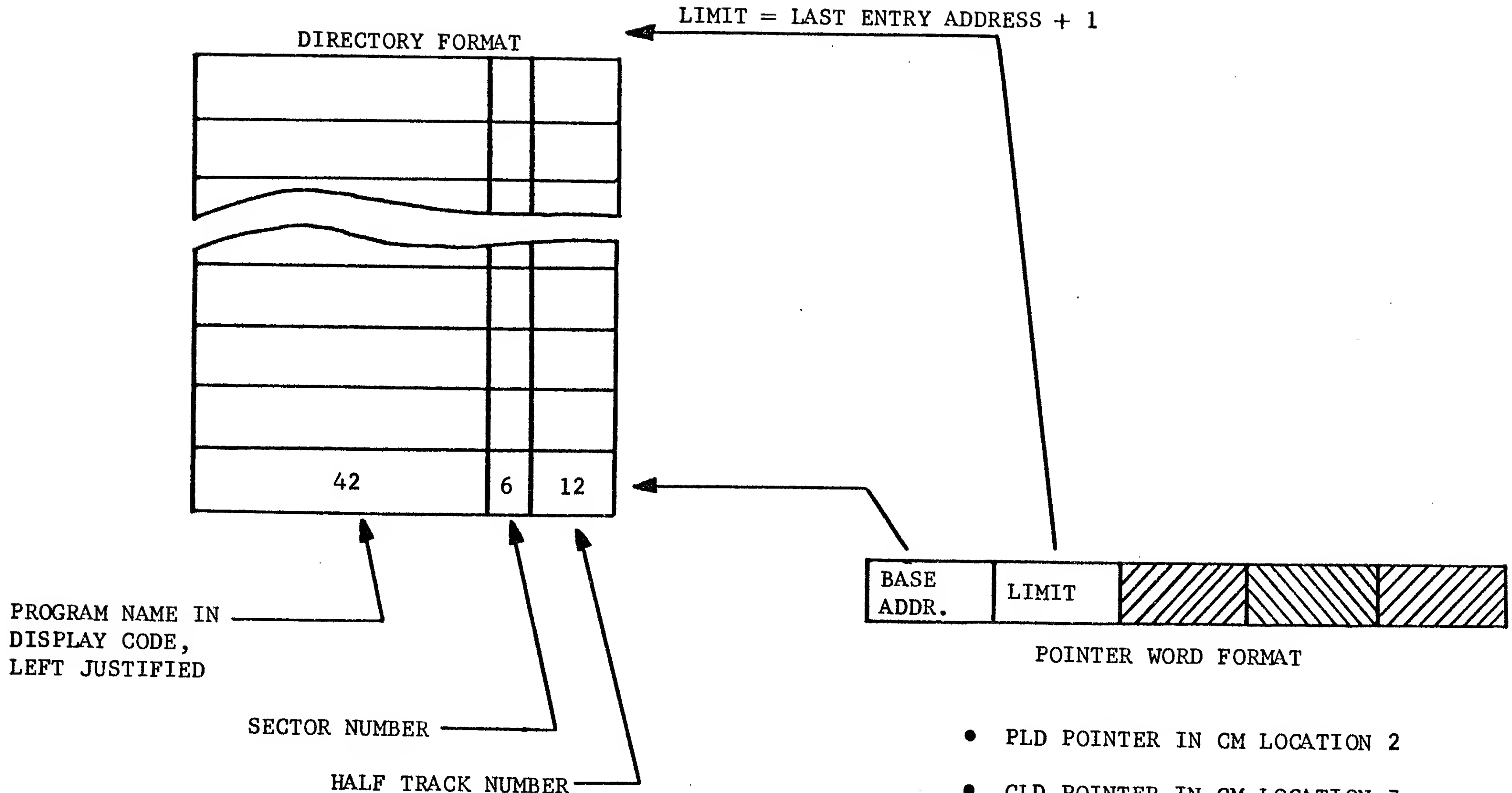
## CM RESIDENT: DISK LIBRARY DIRECTORIES

The central memory resident contains two disk library directories: the Peripheral Library Directory (PLD) for the library of peripheral processor programs on the disk, and the Central Library Directory (CLD) for the library of central processor programs on the disk. The location and size of the Peripheral Library Directory is defined by the PLD pointer word in central memory location 2, and the location and size of the Central Library Directory is defined by the CLD pointer word in central memory location 7. The nominal size of the CLD is $200_8$ locations, while that of the PLD is $100_8$ locations.

The directory format is the same for both the PLD and the CLD, and is illustrated in figure 5. The high-order 42 bits of the directory entry contain the program name in display code, left-justified. The next six bits contain the sector number of the first disk sector for this program, while the low-order 12 bits give the half track number for this program. The program may occupy one or more sectors on the disk: the end of the program is indicated by a short sector (a sector of less than $100_8$ central memory words).

Byte one (the leftmost byte) of the pointer word supplies the base address of the directory: byte two supplies the directory limit address, which is the address + 1 of the last directory entry. When the directory is being searched, exit from the search occurs (in the non-hit case) when the limit is reached or, in some cases, when an entry with byte one equal to zero is detected. If it is desired to delete an entry temporarily for some reason, then, the entry should be set to something other than zero.

The PLD is searched by the peripheral processor resident programs and by certain of the transient programs and overlays. When a peripheral processor resident is directed by MTR to load and execute a peripheral processor program, it first searches the central memory Resident Peripheral Library (RPL) for that program: if the program is not found in the resident library, the peripheral




- PLD POINTER IN CM LOCATION 2
- CLD POINTER IN CM LOCATION 7

Figure 5

DISK LIBRARY DIRECTORIES - PLD & CLD

processor resident proceeds to search PLD. It is thus possible to reduce the size of the resident library by placing some of the peripheral processor programs on the disk. Some peripheral processor transient programs follow the same procedure in loading their overlays: others, however, search only the resident library. It is therefore not possible to move all peripheral processor programs from the resident library to the disk library.

The CLD is searched by two programs: the Central Library Loader (CLL) and the control statement translator (2TS). The function of the Central Library Loader is to load overlays into central memory when called by a central processor program. CLL first searches the central memory Resident Subroutine Library (RSL) for the requested overlay: if not found, the CLD is then searched. If the overlay is not found in either the resident library or the disk library, CLL then searches the File Name Table (FNT) for a file with this name. The control statement translator, 2TS, searches CLD when processing program cards. When the statement translator finds a program card, it first searches the File Name Table for a file with that name: if not found, CLD is searched next. If the program is not found in either the FNT or the CLD, a search is made of the Peripheral Library Directory.

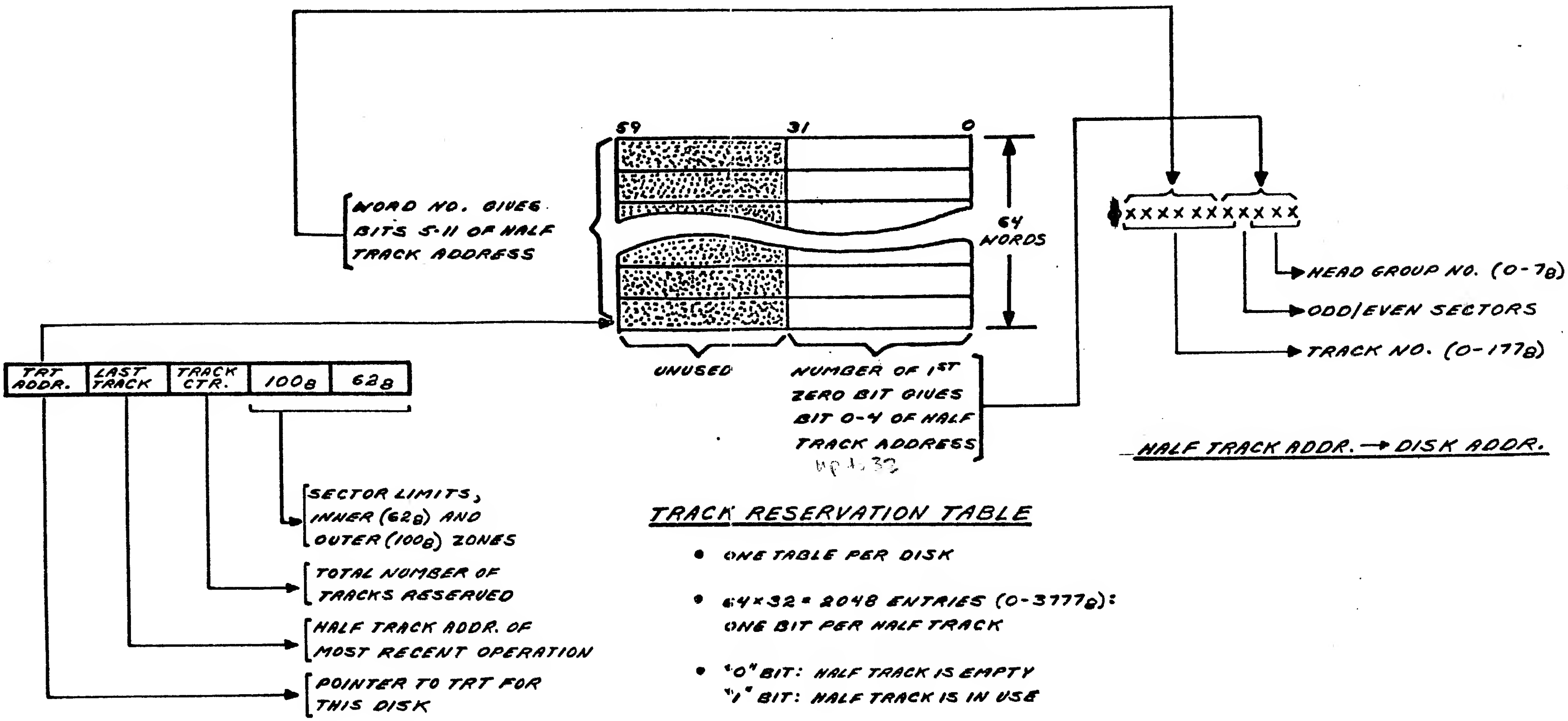
## CM RESIDENT: THE TRACK RESERVATION TABLES

The Chippewa Operating System is designed to permit the use of up to three 6603 diskfiles with the system: these diskfiles are identified as Disk 0, Disk 1, and Disk 2. Both the system and the user may store data on Disk 0, while Disk 1 and Disk 2 are reserved soley for the user. The utilization of space on a given diskfile is recorded in a table called the Track Reservation Table (TRT). There is a Track Reservation Table for Disk 0, for Disk 1, and for Disk 2: the locations of these tables are given by the TRT pointer words in central memory locations 10, 11, and $12_8$, respectively. The tables are identical and are identically manipulated.

Since a single peripheral processor cannot maintain a continuous flow of data between a diskfile and central memory, the Chippewa Operating System employs an interlacing scheme in which data is recorded on only the odd-numbered sectors or only the even-numbered sectors in a track during a revolution over that track. From a hardware standpoint, a track contains either 128 or 100 sectors, depending upon whether the track lies in the two outer zones or the two inner zones. The Chippewa Operating System considers a physical track to be composed of two half tracks; one consisting of the odd-numbered sectors on the physical track, the other consisting of the even-numbered sectors on that track. A half track, then, contains either $64_{10}$ or $50_{10}$ sectors, depending upon its location. Since a diskfile contains 128 tracks at each of 8 head group selections (1024 tracks), a diskfile contains 2048 ($3777_8$) half tracks. A given half track is never used for records of more than a single file: should a file consist of only a single sector, an entire half track would be reserved for that file.

The Track Reservation Table is illustrated in figure 6. The table is made up of 64 words: only the rightmost 32 bits in a word are used. The table thus contains 2048 bits, one bit for each half track on a diskfile. If a bit is zero, the corresponding half track is not in use. If a bit is one, the corresponding half track has been assigned. Should a section of the diskfile become defective, the corresponding bit or bits in the TRT may be permanently set to one (by modifying the library tape) in order to avoid accessing the defective areas.

Half track assignments are handled by MTR. When MTR receives a track request from a peripheral processor program, it searches the TRT until the first zero bit is found. The coordinates of this bit are then assembled to form a half track number: the bit position in the word ($0$ - $37_8$) comprises the low




TRT POINTER WORD FORMAT

(CM LOC. $10_8$ FOR DISK 0)

**TRACK RESERVATION & ADDRESSING**

CHIPPEWA OPERATING SYSTEM

Figure 6

| | | | |
|---|---|---|---|
| CONTROL DATA CORPORATION SOFTWARE DOCUMENT | DOCUMENT CLASS TRAINING; MACH TYPE 6000 | PROJECT NO | DOCUMENT ABSTRACT |
| SAMPLE CODE ☐ | DOCUMENT TITLE TRACK RES. & ADDR. | PROJECT MGR | |
| FLOWCHART ☐ | PAGE OF | PROJECT NAME | |
| DECISION TABLE ☐ | NUMBER; ISSUE DATE | TASK NO | |
| OTHER ☒ | DRAWN BY MHM; DATE 9/13 | TASK NAME | |

order five bits, while the word position in the table (0 - $77_8$) provides the next six bits. MTR returns this half track number to the requesting processor and sets the bit in the table to one. Dropping of track assignments takes place in a reverse fashion. To drop a half track assignment, the requesting processor sends MTR the number of the half track to be dropped. MTR disassembles the half track number into table coordinates and clears the bit in the table.

The low-order three bits of the half track number specify the head group; the next bit ($2^3$) specifies whether this half track uses the odd-numbered sectors ($2^3 = 1$) or the even-numbered sectors ($2^3 = 0$); the next seven bits specify the track number. Since the lower portion of the half track number comes from the bit position in a table word, the order of selection is such that the even-numbered half tracks at head groups 0 - 7 are selected first, and the odd-numbered half tracks at head groups 0 - 7 are selected next. Only when all the half tracks at a given physical position of the heads have been assigned is a half track number selected which requires repositioning. Thus, the layout of the table eliminates unnecessary repositioning.

Byte 1 of the TRT pointer word contains the base address of the table. Byte 2 contains the last half track used by this diskfile, and thus reflects the current physical position of the heads and the currently selected head group. Whenever a disk operation is initiated, the half track number for the operation is compared with the contents of byte 2, and repositioning or head group selection performed only if necessary. This byte is updated at the end of each disk operation.

Bytes 4 and 5 of the pointer word always contain the constants $100_8$ and $64_8$, respectively. These constants are the sector limits for tracks in the outer zones (hig-order bit of the head group number = 0) and the inner zones (high-order bit of the head group number = 1). The disk routines compare the sector number

for the current operation with the appropriate one of these two constants in order to determine when the end of a half track has been reached.

## CM RESIDENT: FILE NAME TABLE/FILE STATUS TABLE

The various types of files currently being controlled by the system are defined by entries in the File Name Table/File Status Table. These two tables are interleaved such that the File Name Table (FNT) entry and the File Status Table (FST) entry for a specific file occupy successive central memory locations. The base address and limit address (last entry address + 1) of the FNT/FST are contained in bytes 1 and 2, respectively, of the FNT/FST pointer word in central memory location 4. The nominal size of the FNT/FST is $1000_8$ central memory words, permitting up to $256_{10}$ files to be defined at any one time.

The format of the FNT/FST entry is shown in figure 7. The FNT entry contains the file name in display code in the leftmost 42 bits of the word. The next six bits contain the priority, if any, associated with this file. The low-order six bits of the entry contain a File Type indicator (3 bits) and the Control Point Number (3 bits) to which this file is assigned: if unassigned, the Control Point Number is zero. The File Type indicator may take on the values 0, 1, 2, or 3, indicating that this file is, respectively, an INPUT file, an OUTPUT file, a COMMON file, or a LOCAL file.

When a job enters the system (either from a card reader or from a tape unit), the File Type indicator is set to 0 (INPUT file), the file name is set to the job name as given on the job card, and the priority is entered from the job card. Unassigned (Control Point Number = 0) files on the disk of type INPUT, then, constitute a job stack, and the FNT serves as a job table. When the system is ready to bring in the next job from the disk, it searches the FNT table for the highest priority unassigned INPUT file. When this file is assigned to a control point, the number of this control point is set in the low-order three bits of the FNT entry, and the File Type indicator is set to LOCAL. The job name (file name of an INPUT file) and priority are placed in the control point

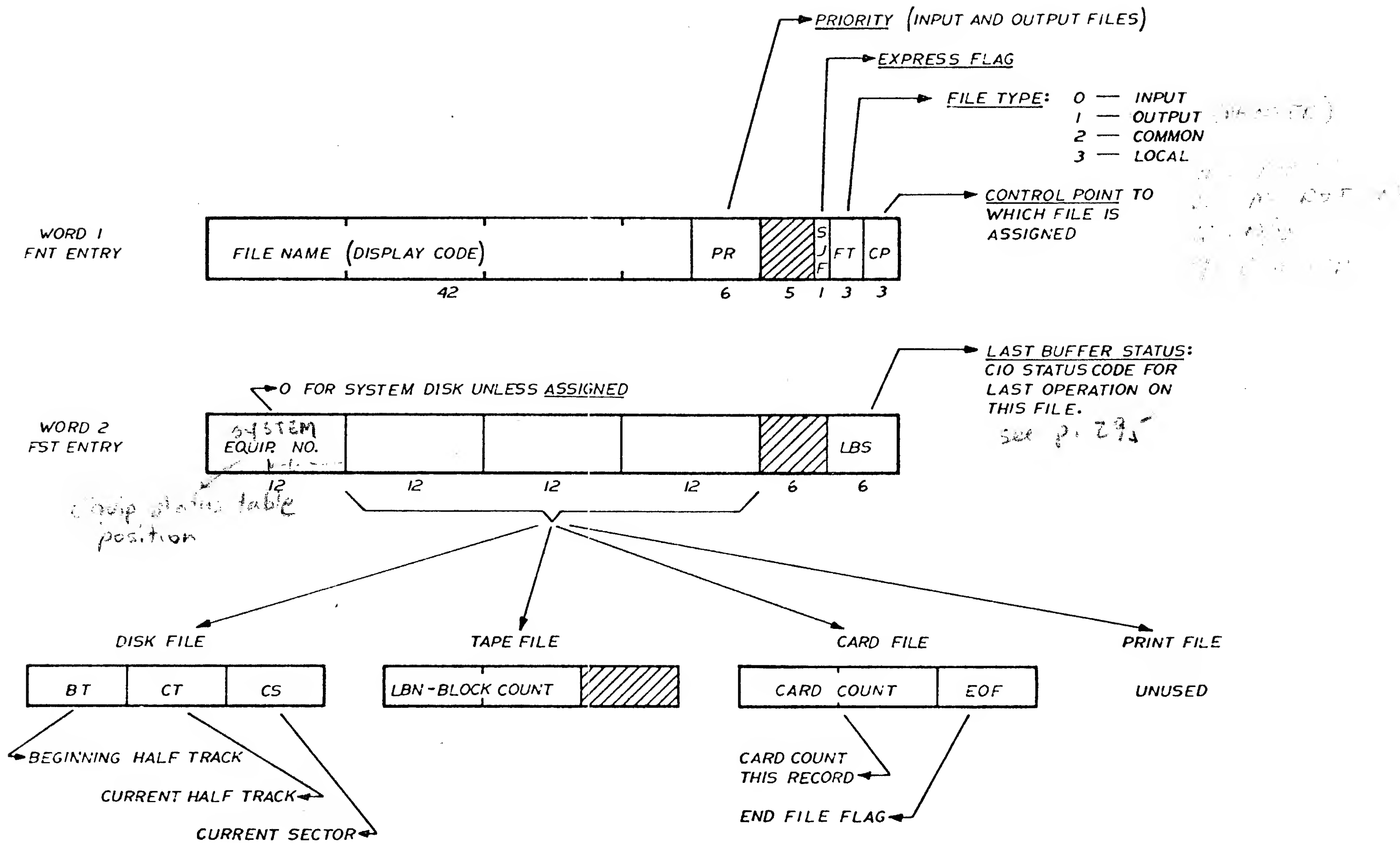
EXPRESS CONTROL POINTS

By use of the DSD keyboard entry n.ONEX, a control point (i.e., control point n) may be designated as an express control point. When DSD detects this entry, it sets an express flag in the high-order bit of word 24 in the control point area, and calls 1BJ to the control point.

When jobs are loaded on the disk by 1LJ, jobs with a time limit of one minute or less and a field length of 40,000 or less are recognized and a flag ($2^6$ bit of byte 5) set in the FNT entry for the job. When 1BJ is brought to an express control point, it searches the FNT only for unassigned files of type INPUT in which this flag is set. If a file of this type is found, it is loaded in the conventional manner. If no file of this type can be found, 1BJ enters the job name "EXPRESS" (rather than "NEXT") in the control point area and enters PP recall.

It is advisable that the high number control points be designated as express control points. Since jobs presumably will be shuttled in and out of express control points at a rate faster than jobs assigned to other control points, express control points will be the source of storage requests more often than other control points. By designating a high-numbered control point (e.g., control point 7) as an express control point, storage allocation can be handled more efficiently.

Express jobs whose priority is higher than other jobs in the FNT will be loaded at any available control point, regardless of whether or not the control point is designated as an express control point.




FNT/FST ENTRY

Figure 7

area. The file name is then set to INPUT, and the priority field cleared. Files may be initiated by a job: if a CIO call specifies a file name which is not contained in the FNT, a new entry is added to the FNT which contains the specified file name, has the file type LOCAL, and is assigned to the disk.

Data to be printed at the end of a job is written by the job to a LOCAL file on the disk with the file name OUTPUT. At the end of the job, all LOCAL files except the LOCAL file named OUTPUT are dropped. The system routine which closes out a job (1AJ) changes the name of this file from OUTPUT to the job name, changes the type from LOCAL to OUTPUT, and enters the priority from the control point area. Effectively, then, files on the disk of type OUTPUT constitute a job stack for the print package. The print package selects the next file to be printed by searching the FNT for the file of type OUTPUT with the highest priority.

If it is desired to retain a file at the end of a job for use with some subsequent job, the file must be declared type COMMON by means of a COMMON control card. At the end of a job, a file of type COMMON will not be dropped: the control point assignment will simply be cleared. COMMON type files may be dropped when desired by use of a RELEASE control card.

The format of the FST entry varies, depending upon the type of equipment assigned for the file. Files are assigned to disk 0 unless another equipment is specified by means of an ASSIGN control card. Byte 1 of the FST entry always contains the equipment number, which gives the relative location in the Equipment Status Table of the equipment type for this file. This byte is either set by the statement translator (2TS) when an ASSIGN control card is processed or, if no ASSIGN card appears for this file, is set to correspond to disk 0 when the first reference to this file is made (by the 2BP overlay). Byte 5 of the FST entry contains the buffer status: this is obtained from the CIO call and inserted in the FST entry (by 2BP) for use by the various I/O routines: this status indicates the type of operation to be performed (read, write, rewind, etc.). If

the $2^0$ bit of this byte is zero, an operation involving this file is in process: if the $2^0$ bit is one, this file is not reserved.

Bytes 2, 3, and 4 of the FST entry vary according to the equipment type. In the case of the printer, these bytes are not used. In the case of the card reader, bytes 2 and 3 are used to maintain a count of the number of cards processed in a record, and byte 4 is set when an end-of-file card (6-7-8-9 card) is processed. For tape files, bytes 2 and 3 are used to maintain a count of the number of blocks recorded for this file.

For disk files, byte 2 holds the beginning half track number for the file, byte 3 holds the current half track number (i.e., the half track on which the most recent operation involving this file took place) for this file, and byte 4 holds the current sector number. The next read or write to this file will be to the sector supplied by byte 4 on the half track supplied by byte 3. When housekeeping for this read or write is performed, the current half track number in byte 3 will be compared with the last half track byte in the TRT pointer word to determine if repositioning and/or head group selection is necessary.

When a file assigned to the disk is rewound, the current half track byte is set equal to the beginning half track byte and the current sector number is set to zero. Disk files which are COMMON type files are not rewound at the end of the job.

CM RESIDENT: DAYFILE BUFFER

The dayfile contains a variety of information concerning the status and progress of jobs in the system, such as start and finish times, peripheral and central processor usage, diagnostics, etc. Dayfile messages may be issued by any of the system peripheral processor programs and may also be issued by a user's central processor program via the MSG routine.

The dayfile is maintained on the disk: dayfile entries are buffered through a portion of the central memory resident area called the Dayfile Buffer. The

base address and limit address (last word address + 1) of this buffer are supplied by the DFB pointer word in central memory location 3. The nominal size of the Dayfile Buffer is $1000_8$ locations.

The Dayfile Buffer and its pointer word are illustrated in figure 8. The first four bytes in the pointer word contain the base address, IN pointer, OUT pointer, and the limit address: these quantities are analogous to the FIRST, IN, OUT, and LIMIT pointers used in CIO, and the Dayfile buffer is handled in much the same manner as a CIO processed buffer.

When a peripheral processor program wishes to insert a message in the dayfile, it places the message in its Message Buffer and issues the appropriate request to MTR. MTR copies the message from the Message Buffer into the Last Dayfile Message area in the control point area of the job to which the requesting processor is assigned. MTR then enters the message, together with the job name and the time, in the Dayfile Buffer beginning at the location specified by the IN pointer byte of the DFB pointer word.

Whenever MTR enters a message in the Dayfile Buffer, a test is made to determine if the buffer contains a full sector of data. (It is possible that the message just entered resulted in the buffer's containing slightly more than a full sector.) If it does, a flag is set which causes the full sector and the partial sector, if any, to be dumped to the disk. Dumping is done by MTR in six phases in order to avoid tying up MTR for an extended period of time. After each phase has been executed, MTR returns to its master loop to perform any functions required by other peripheral processors or the central processor. As data is transferred from the buffer to the disk, the OUT pointer is adjusted accordingly. Insofar as maintaining the dayfile on the disk is concerned, only slightly more than $100_8$ words are required. The nominal size of the Dayfile Buffer is set at $1000_8$ words to permit DSD to display as much dayfile activity as possible. Thus, if the buffer size is reduced to about $110_8$ words, the sole effect is to reduce the size of the dayfile console display.

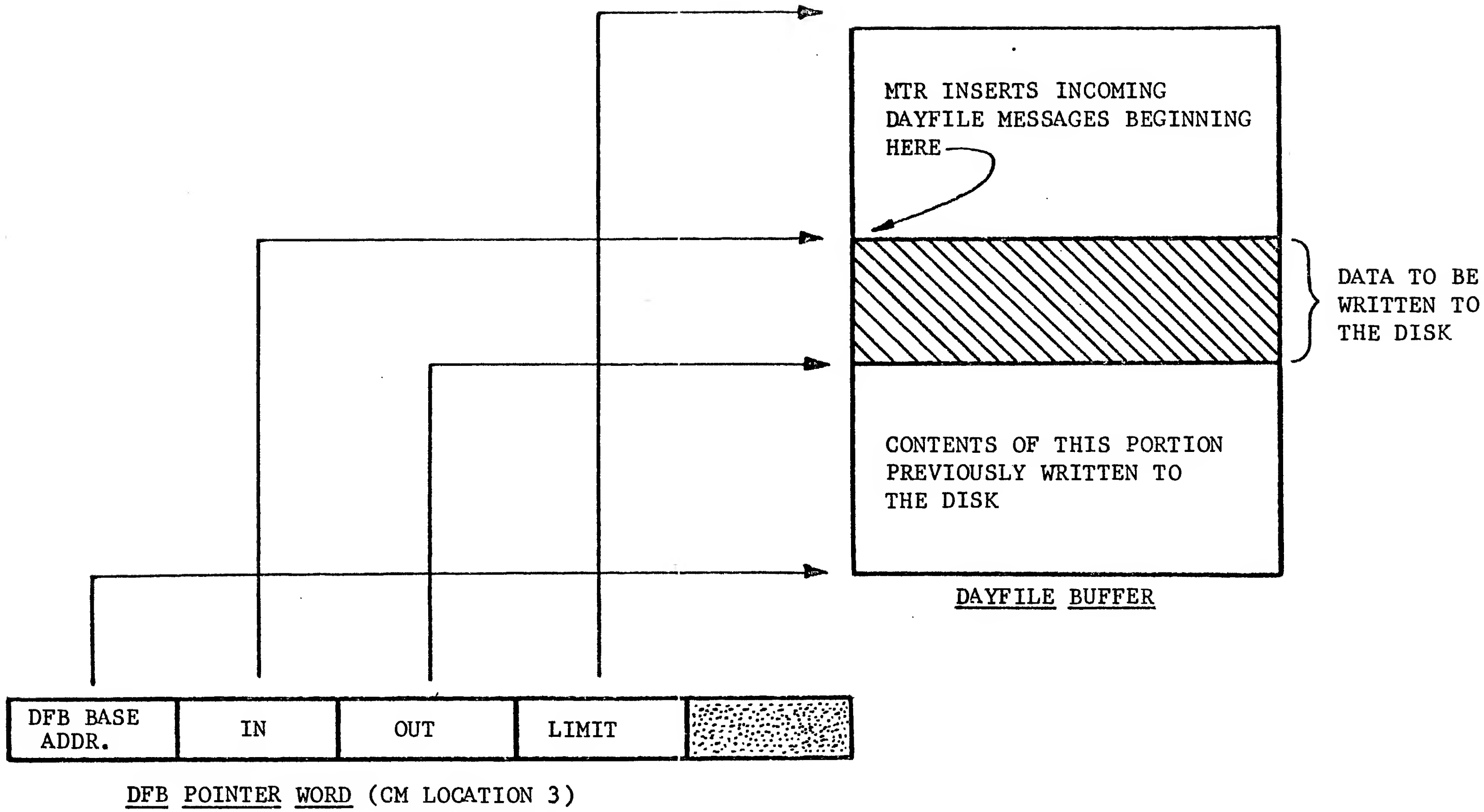



Figure 8

DAYFILE BUFFER

CM RESIDENT: RESIDENT LIBRARIES

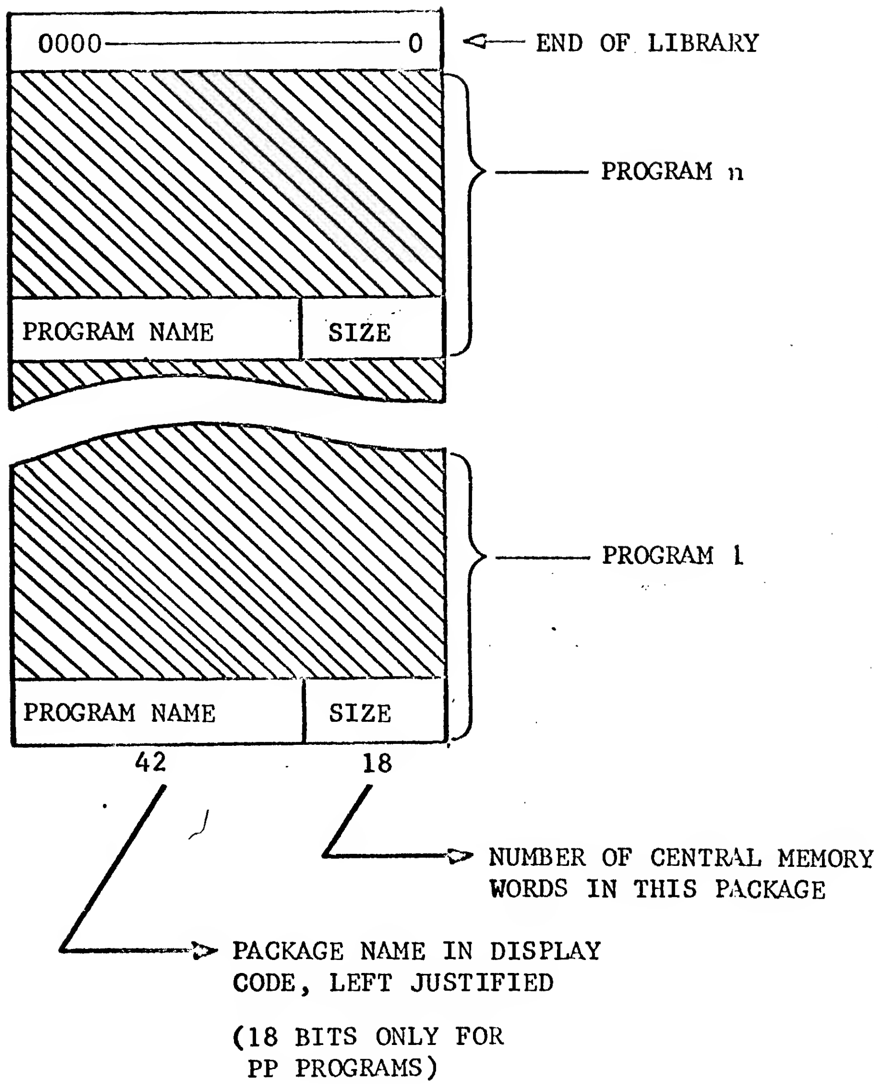
The central memory resident contains two libraries: the Resident Peripheral Library (RPL), which contains peripheral processor programs such as 1AJ, 2RD, and CIO, and the Resident Subroutine Library, which contains central programs such as ACOS and TIME. The starting addresses of the Resident Peripheral Library and the Resident Subroutine Library are defined by the RPL and RSL pointer words in central memory locations 1 and 6, respectively.

The library format is the same for both RSL and RPL, and is illustrated in figure 9. The first word of a library program contains the name in display code in the leftmost 42 bits. The low-order 18 bits of the word contain the package size in central memory words. In searching the library, the searching routine reads the program name of the first package and tests to see if this is the desired routine: if it is not, the size of the routine is added to the base address to form the address of the first word of the next program. It is important then, that the size value be correct. The end of each library is indicated by a word of zeroes.

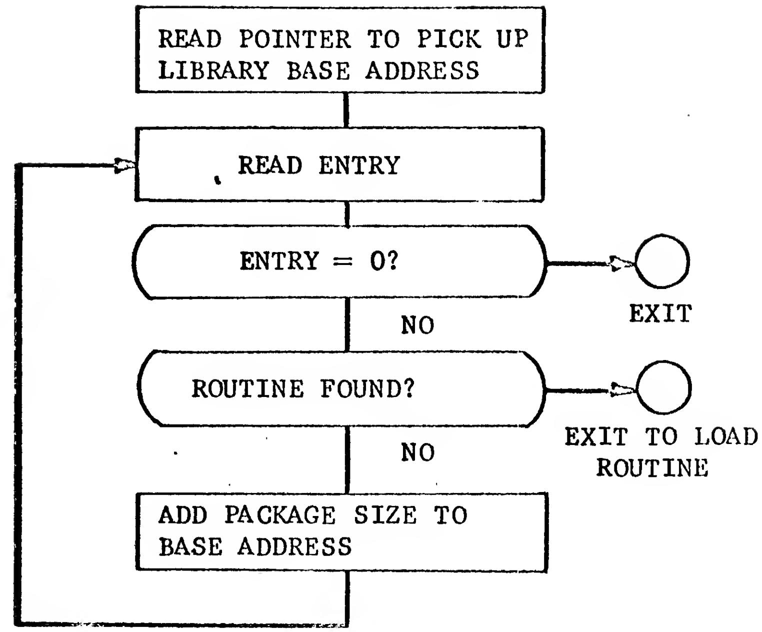
The RPL is searched by the peripheral processor resident programs and by most of the transient programs. If the peripheral processor resident does not find a routine in the RPL, it proceeds to search the PLD. Transient programs such as 1AJ, 1BJ, CIO, etc., are loaded into peripheral processor memory beginning at location $773_8$: the first executable instruction, which is in the first byte of the second central memory word in the package, is thus at location $1000_8$. Overlay programs, such as 2RD, 2BP, etc., are loaded into peripheral processor memory beginning at location $1773_8$. Since these programs are entered via a return jump (to location $2001_8$), the first executable instruction is at location $2002_8$, with location $2000_8$ containing the LJM order code for the exit point.

The RSL is searched by the CLL (Central Library Loader) routine. Programs in the RSL are assembled to execute beginning at location 0, and so must be

LIBRARY FORMAT




RPL/RSL SEARCH




*RESIDENT LIBRARIES*

Figure 9

relocated by the user to the desired location.

The size of the RSL and RPL can be reduced by transferring programs to the disk libraries. Certain programs, however, may not be transferred, since not all peripheral processor transient programs search the PLD if a routine is not found in the RPL. For example, system programs such as 1AJ, 1BJ, and 1DJ search only the RPL for their overlays (the transient programs themselves, however, could be transferred to the disk library). Other transient programs, such as CIO, search both RPL and PLD for overlays.

CONTROL DATA
CORPORATION
8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

PRINTED IN U.S.A.

# CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

## CHIPPEWA OPERATING SYSTEM DOCUMENTATION

Volume II Preliminary Edition

CONTROL DATA CORPORATION

Development Division - Applications

# SYSTEM PERIPHERAL PACKAGES AND OVERLAYS

Chippewa Operating System

10/21/65

Table of Contents

# SYSTEM PERIPHERAL PACKAGES AND OVERLAYS

## INTRODUCTION

All peripheral packages that begin with a numberal are special operating system packages or equipment driver overlays. The system packages begin with the numeral "1" and begin execution at address 1000 of peripheral memory. Their functions are to load jobs onto the disk, make control point assignments, process the control statements, and print the jobs' output. Whenever specialized operations , i.e. read tape, punch cards, translate control statements, etc., are required, an overlay is loaded into the requesting PP at location 2000. These overlays begin with the numberal "2" and parameters are passed to them by direct core cells ($1\text{-}74_8$). Most of them are maintained in RPL (resident peripheral library), however they could be kept in PLD (peripheral library directory) if the system packages searched this table. Since most of them are fairly short, the system packages expect them to reside in central memory.

ROUTINE: 1AJ - Advance Job.

PURPOSE: To advance the status of a job by controlling the processing of the next control card or terminating the job.

GENERAL: This package is called by MTR on its main loop and the following conditions prevail when 1AJ is called.

a. A job has been assigned to a control point by 1BJ.

b. The central processor is not executing the job at the control point.

c. The storage move flag is not set.

d. The control point is not listed in the CPU stack, i.e., it is not waiting on the central processor.

METHOD:

1. If an error flag for the control point is set, 2EF is called to process the error. This routine will issue the proper error diagnostic to the dayfile and then position the control card buffer parameters to the statement after an EXIT card or, if no EXIT card to found, to the record separator.

2. 2TS is called to process the control statements in the order encountered and all the statements will be processed before 1AJ regains control.

3. If the control point has zero priority, i.e., PP program that uses central memory, all files and equipment assigned to this control point are dropped by 2DF and monitor. A request is also made to monitor to release the storage reserved by this control point and a pause loop is maintained until the field length is zero. The control point is then cleared of information and 1AJ is released. No dayfile data will be written in this case.

4. In the normal case with a priority set at the control point, an attempt is made to locate an "OUTPUT" circular buffer so that it may be emptied if it is not. The first $100_8$ words of the program are searched for the buffer. The lower 18 bits of each of these words specify an address where the file name and status is located. If the address is within the field length, the name is checked for "OUTPUT." The search continues until $RA+100_8$ words have been checked.

5. If the buffer status indicates that a file mark has already been requested, it is assumed that the buffer is emptied of usable information. If the file mark is not set, then the buffer will be dumped if

   a. it is a disk file

   b. the last operation was a write.

   2WD is called to write the buffer contents on the disk.

6. Both the amount of central processor and peripheral processor running time is read from the control point, converted to decimal, and sent to the dayfile.

7. A search is made of FNT to find a file named "OUTPUT" assigned to this control point. If there is none, then such a name is entered into the FNT so that the dayfile can be printed.

8. The file name is then changed to that of the job name and the job's priority is also put into the FNT. The file is released from the control point by putting a zero value in the control point byte. This action will cause the print routines (1DJ or 1TD) to sense a file ready for printing.

9. All files assigned to this control point in the FNT are dropped by 2DF. The FNT/FST entries are completely zeroed.

10. The upper most byte of the EST has the control point assignment for the equipment. All the pieces of equipment assigned by this job are released by a monitor request.

11. The control point area is then cleared and 1BJ is called to this PP so that another job may be assigned.

## 1AJ ROUTINES

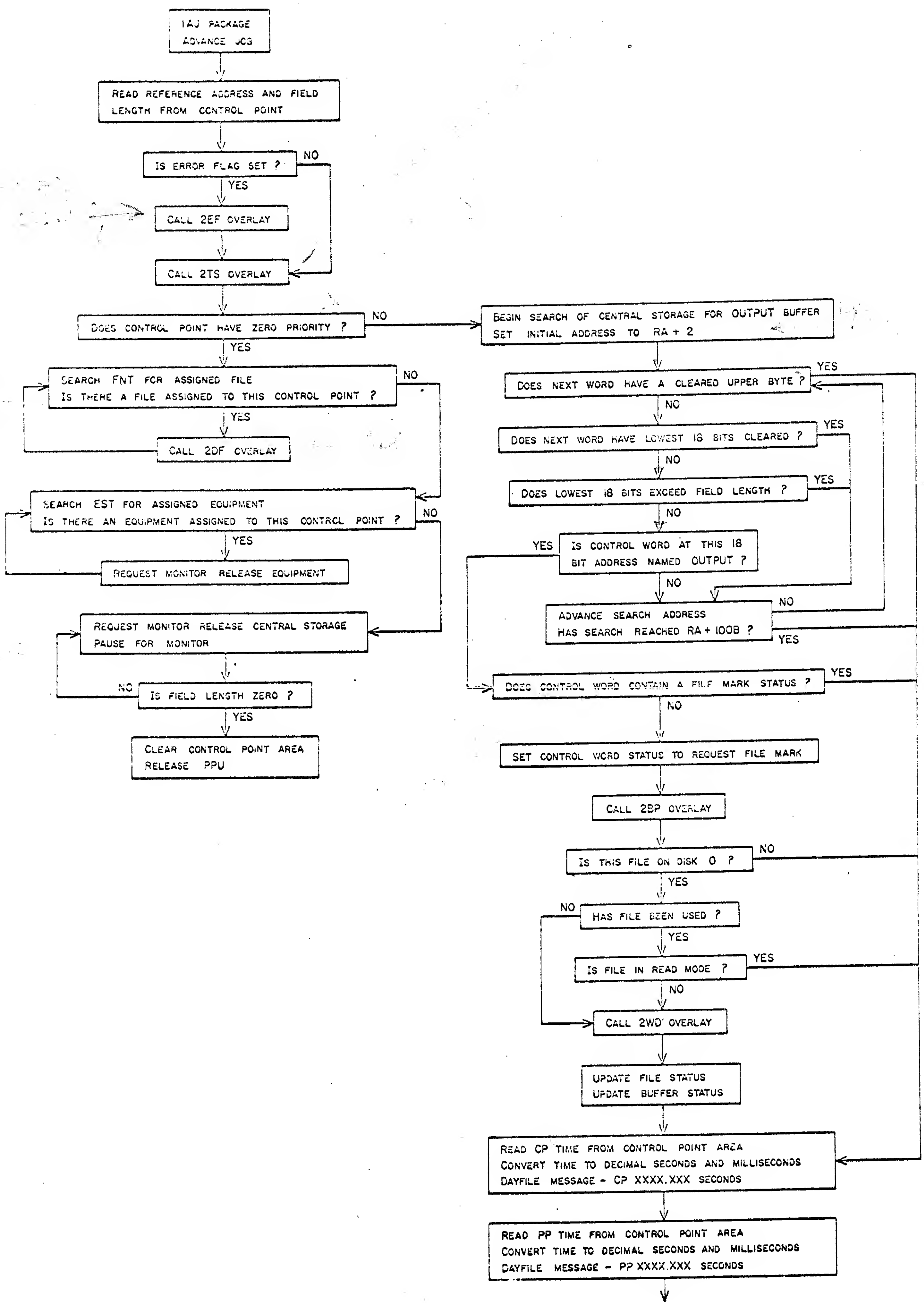
| | | |
|---|---|---|
| 1000 | MAIN PROGRAM | 1700, 1410, 1740, 1500<br>1100, 1320, 100, 12-760 |
| 1100 | RECORD RUNNING TIMES | 1200, 530, 530 |
| 1200 | DECIMAL CONVERSION | |
| 1320 | RELEASE OUTPUT FILE | 1640 |
| 1410 | DROP FILES | 1700, 23-760 |
| 1500 | SEARCH FOR OUTPUT BUFFER | 1700 |
| 1640 | BEGIN OUTPUT FILE | 740, 750 |
| 1700 | CALL SUBROUTINE | 2000 |
| 1740 | CLEAR CP AREA | 10-760, 17-760 |

## DIRECT CORE CELLS

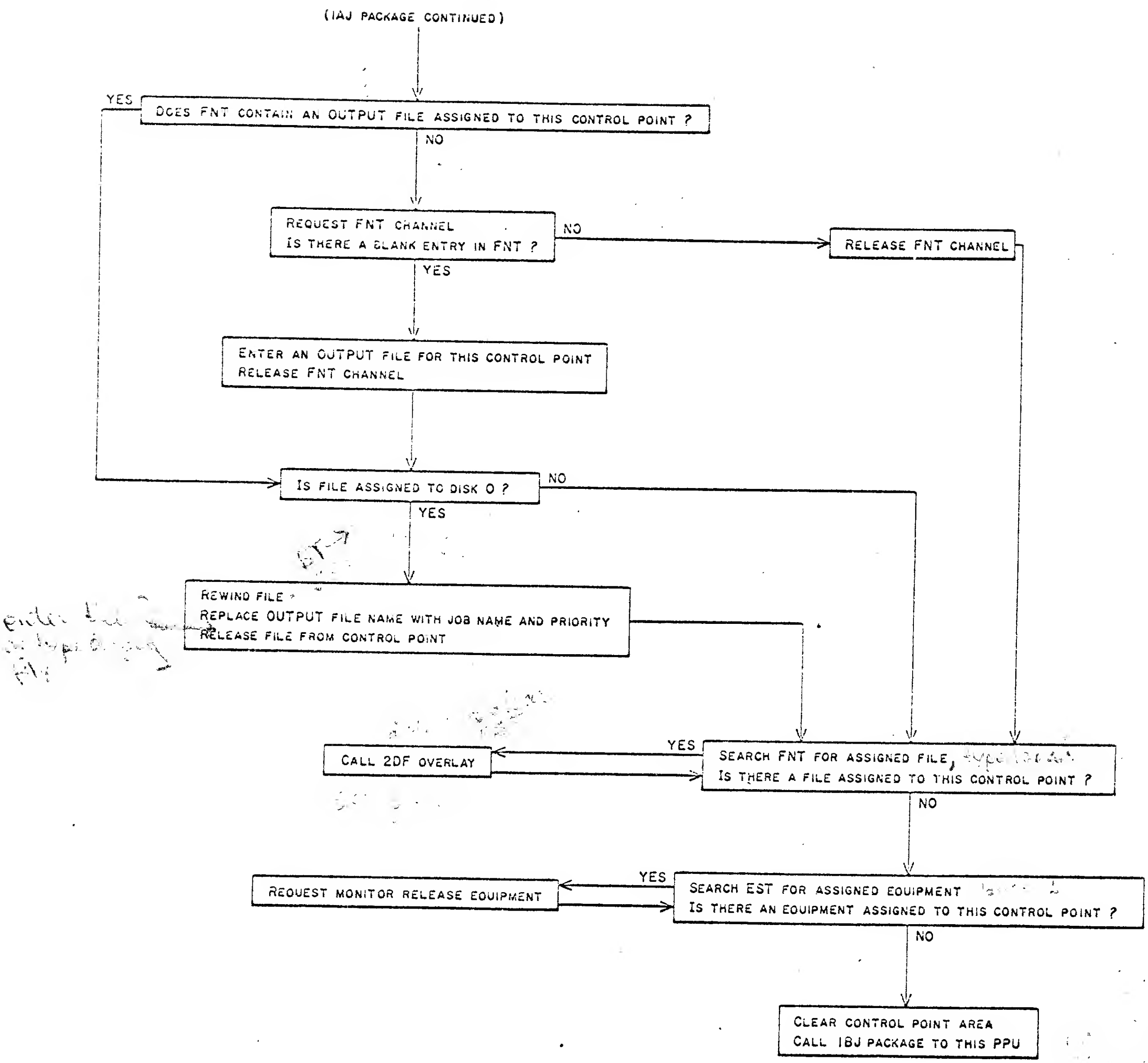
| | | |
|---|---|---|
| 1000 | P10/14 | CP STATUS |
| | P55 | RA |
| | P56 | FL |
| | P50/54 | CONTENTS OF INPUT REGISTER |
| | P70 | CONSTANT 1 |
| | P71 | CONSTANT 100 |
| | P72 | CONSTANT 1800 |
| | P74 | CP ADDRESS |
| | P75 | ADDRESS OF INPUT REGISTER |
| 1100 | P01 | MESSAGE WORD COUNT |
| | P10/14 | CPTIME, LATER PP TIME |
| | P20/30 | CP TIME MESSAGE, LATER PP TIME MESSAGE |
| | P74 | CP ADDRESS |
| 1200 | P10/14 | CP OR PP TIME |

| | | |
|---|---|---|
| | P20/30 | CP TIME MESSAGE. LATER PP TIME MESSAGE |
| | P71 | CONSTANT 100 |
| 1320 | P10/14 | FNT ENTRY |
| | P20/24 | FNT STATUS |
| | P40/44 | CP(21) WITH ADDED PRIORITY |
| | P50/54 | INPUT REGISTER |
| | P74 | CP ADDRESS |
| 1410 | P10/14 | FNT STATUS, LATER EST ENTRY |
| | P20/24 | EST STATUS |
| | P40/44 | FNT ENTRY |
| | P46 | FIRST OF FNT |
| | P47 | IN OF FNT |
| | P50/54 | INPUT REGISTER |
| | P74 | CP ADDRESS |
| 1500 | P01 | SEARCH ADDRESS |
| | P10/14 | ARGUMENTS LOCATED AFTER RA+2 |
| | P20/24 | CONTROL WORD OF ARGUMENT AT RA+2+n |
| | P40/44 | BUFFER STATUS |
| | P45 | LAST BYTE OF CONTROL WORD |
| | P54 | RA |
| | P55 | FL |
| | P57 | FST ADDRESS |
| 1640 | P01 | CONSTANT 2 |
| | P10/14 | FNT ENTRY |
| | P20/24 | FNT STATUS |
| 1700 | (A) | SUBROUTINE NAME |
| | P01 | RPL INDEX |
| | P02/03 | SUBROUTINE NAME |

| | | |
|---|---|---|
| | P04 | RPL STARTING ADDRESS |
| | P10/14 | RPL ENTRY |
| 1740 | P01 | MAXIMUM $200_8$ WORD COUNTER |
| | P10/14 | CP STATUS, LATER ZERO WORD |


IAJ PACKAGE
ADVANCE JOB
READ REFERENCE ADDRESS AND FIELD
LENGTH FROM CONTROL POINT
IS ERROR FLAG SET ?
NO
YES
CALL 2EF OVERLAY
CALL 2TS OVERLAY
DOES CONTROL POINT HAVE ZERO PRIORITY ?
NO
YES
SEARCH FNT FOR ASSIGNED FILE
IS THERE A FILE ASSIGNED TO THIS CONTROL POINT ?
CALL 2DF OVERLAY
SEARCH EST FOR ASSIGNED EQUIPMENT
IS THERE AN EQUIPMENT ASSIGNED TO THIS CONTROL POINT ?
REQUEST MONITOR RELEASE EQUIPMENT
REQUEST MONITOR RELEASE CENTRAL STORAGE
PAUSE FOR MONITOR
IS FIELD LENGTH ZERO ?
CLEAR CONTROL POINT AREA
RELEASE PPU
BEGIN SEARCH OF CENTRAL STORAGE FOR OUTPUT BUFFER
SET INITIAL ADDRESS TO RA + 2
DOES NEXT WORD HAVE A CLEARED UPPER BYTE ?
DOES NEXT WORD HAVE LOWEST 18 BITS CLEARED ?
DOES LOWEST 18 BITS EXCEED FIELD LENGTH ?
IS CONTROL WORD AT THIS 18
BIT ADDRESS NAMED OUTPUT ?
ADVANCE SEARCH ADDRESS
HAS SEARCH REACHED RA + 100B ?
DOES CONTROL WORD CONTAIN A FILE MARK STATUS ?
SET CONTROL WORD STATUS TO REQUEST FILE MARK
CALL 2BP OVERLAY
IS THIS FILE ON DISK 0 ?
HAS FILE BEEN USED ?
IS FILE IN READ MODE ?
CALL 2WD OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
READ CP TIME FROM CONTROL POINT AREA
CONVERT TIME TO DECIMAL SECONDS AND MILLISECONDS
DAYFILE MESSAGE - CP XXXX.XXX SECONDS
READ PP TIME FROM CONTROL POINT AREA
CONVERT TIME TO DECIMAL SECONDS AND MILLISECONDS
DAYFILE MESSAGE - PP XXXX.XXX SECONDS

( NEXT PAGE )


(IAJ PACKAGE CONTINUED)
DOES FNT CONTAIN AN OUTPUT FILE ASSIGNED TO THIS CONTROL POINT ?
YES
NO
REQUEST FNT CHANNEL
IS THERE A BLANK ENTRY IN FNT ?
NO
YES
RELEASE FNT CHANNEL
ENTER AN OUTPUT FILE FOR THIS CONTROL POINT
RELEASE FNT CHANNEL
IS FILE ASSIGNED TO DISK 0 ?
NO
YES
REWIND FILE
REPLACE OUTPUT FILE NAME WITH JOB NAME AND PRIORITY
RELEASE FILE FROM CONTROL POINT
CALL 2DF OVERLAY
YES
SEARCH FNT FOR ASSIGNED FILE,
IS THERE A FILE ASSIGNED TO THIS CONTROL POINT ?
NO
REQUEST MONITOR RELEASE EQUIPMENT
YES
SEARCH EST FOR ASSIGNED EQUIPMENT
IS THERE AN EQUIPMENT ASSIGNED TO THIS CONTROL POINT ?
NO
CLEAR CONTROL POINT AREA
CALL 1BJ PACKAGE TO THIS PPU

ROUTINE: 1BJ -- Begin Job

PURPOSE: To assign a job to a control point and process the job card.

GENERAL: The package is called by DSD where "X.NEXT" is requested and recalled by 1AJ. The control point assignemtn is specified in the input register upon entry to the package.

METHOD:

1. If the error flag in CP(20) is set, the package is released. The error will be processed later by 1AJ and 2EF.

2. If the priority is not zero, this is a recall entry. Otherwise, the following steps occur:

   a. A search is made through FNT for the highest priority file of TYPE 0 (INPUT) and no control point assignment. If none is found, the job name is set to NEXT (for display) and the message IDLE sent to display and the package released in recall status.

   b. If a file was found, the file name in FNT is set as job name, the file name is changed to INPUT, and the TYPE is changed to local. Also, the priority of the job is set in the CP.

3. If the job cards have been loaded, this is a recall entry. Otherwise, the following steps occur:

   a. $300_8$ words of central memory are requested of MTR. If not assigned, the message WAITING FOR STORAGE is sent to display and the package released in recall status.

   b. If 300 words were assigned, the first ten words are set as follows:

| | | | |
|---|---|---|---|
| RA = RA+1 = RA+2 = | 0 | | |
| RA+3 = | INPUT | 10 | File Name and Buffer Status |
| RA+4 = RA+5 = RA+6 = | 0 | 010 | FIRST, IN, OUT |
| RA+7 = | 0 | 0300 | LIMIT |

   c. 2BP is called to verify the parameters and set direct core cells for 2RD.

   d. 2RD is called to read the control statement record into CM.

   e. FST is updated to reflect a completed read. The control statements are moved from the CM buffer to CP control statement buffer using PP locations beginning at 7000 as a transient buffer.

f. CP(21) is set to reflect the reading of the control statements.

g. 2TJ is called to translate and process the job card. The time limit specified on the JOB card is set by MTR.

4. The field length specified on the job card is requested of MTR. If not assigned, the message "WAITING FOR STORAGE" is sent to display and the package released in recall status.

5. If MTR assigned the memory, the job card is issued to the dayfile.

6. Finally, the package is released. The remaining statements will be processed later by 1AJ and 2TS.

NOTES:

All console messages are sent to display by entering the message in CP (30-37). These messages are line 3 of the control point display.

The job card is sent to the dayfile by storing it in the message buffer (address specified by P77) and issuing a F01 request to MTR.

All overlays called by 1BJ must be in RPL since PLD is <u>not</u> searched when calling the overlays. These overlays include 2BP, 2RD, 2TJ.

Two recall flags are used:

a. priority given by CP(22).

b. control cards loaded or not loaded by CP(21).

Three conditions may exist which will cause 1BJ to be released in a recall status. These are:

1. If there exists <u>no</u> unassigned input files in FNT of the TYPE input.

2. If MTR will not assign storage for the buffer to load the control cards into CM.

3. If MTR will not assign storage for the job as specified by FL on the JOB card.

Upon entry to 1BJ, two flags (see above) specify whether this is the initial entry or a recall entry. If it is a recall entry, the flags given above cause the package to skip the areas of code it has executed on a previous call. For example, if the priority given in CP(22) is zero, this is either the initial entry or no unassigned input file was found on the previous entry (same as initial entry). If the priority is not zero, a file has been assigned and the coding to find and assign a file is bypassed. If the job cards have not been

loaded (specified by CP (21 byte 11-0) = 0) they must be loaded into the CP control statement buffer. If they have been loaded, this coding is skipped. If the priority is non-zero and the job cards are loaded, or after these have been done, storage is requested for the job. If not assigned, the package is released in recall status again. Upon next entry all coding will be skipped except this storage request since the priority will be non-zero and the job cards are loaded.

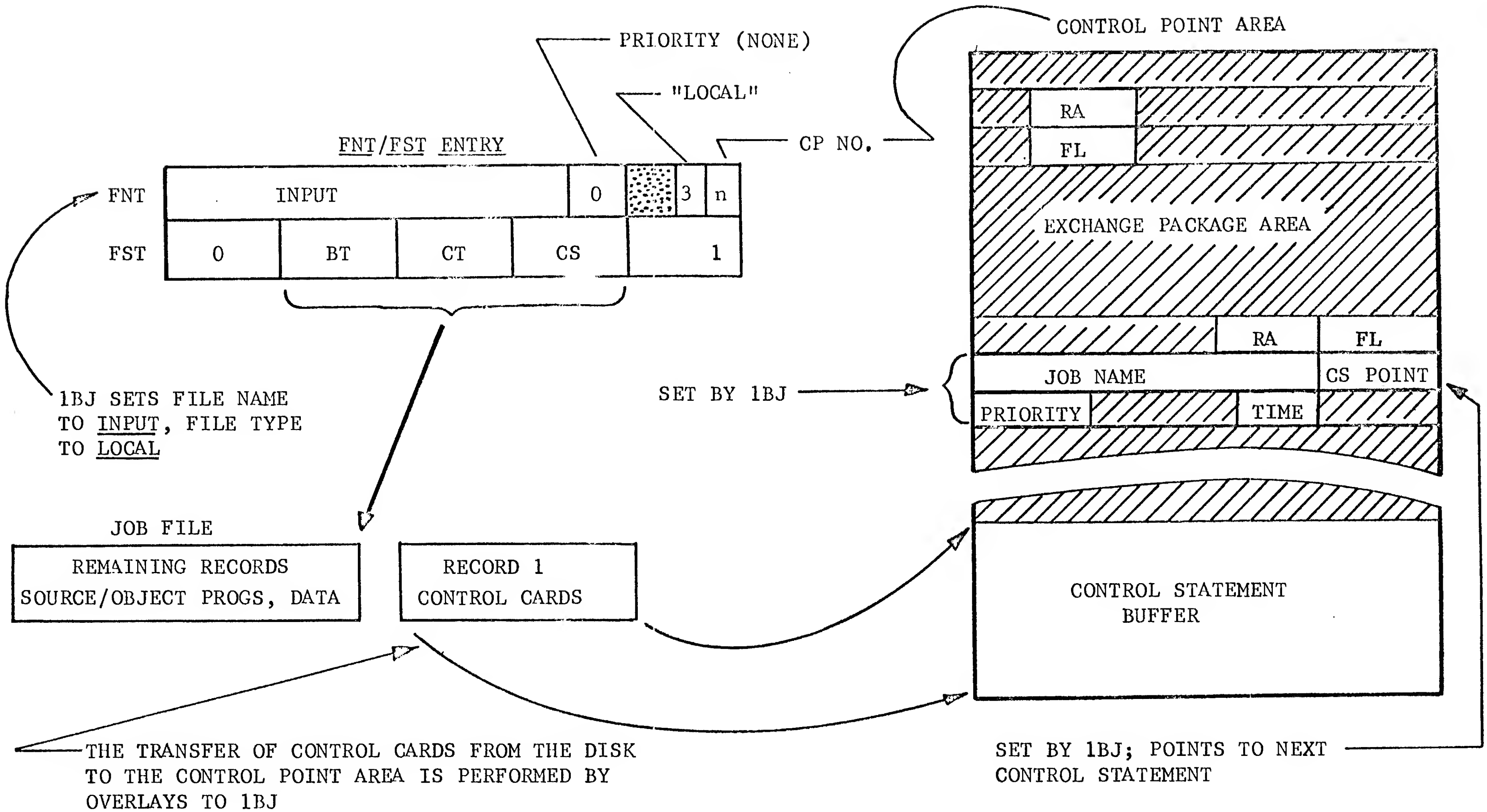
Releasing a PP in recall status involves storing the contents of the input register in CP(25) and then releasing the PP via MTR request 12. A normal release leaves CP(25)=0.

## 1BJ Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1100, 1440, 1400, 1500, 12-760 |
| 1100 | Search for Job | 740, 750, 12-760, 24-760 |
| 1240 | Call (Overlay) Subroutine | |
| 1300 | Read Control Cards | 1240 |
| 1400 | Request Storage | 10-760, 12-760 |
| 1440 | Read Job Cards | 1400, 1300, 1240, 14-760 |
| 1500 | Issue Statement | 01-760 |

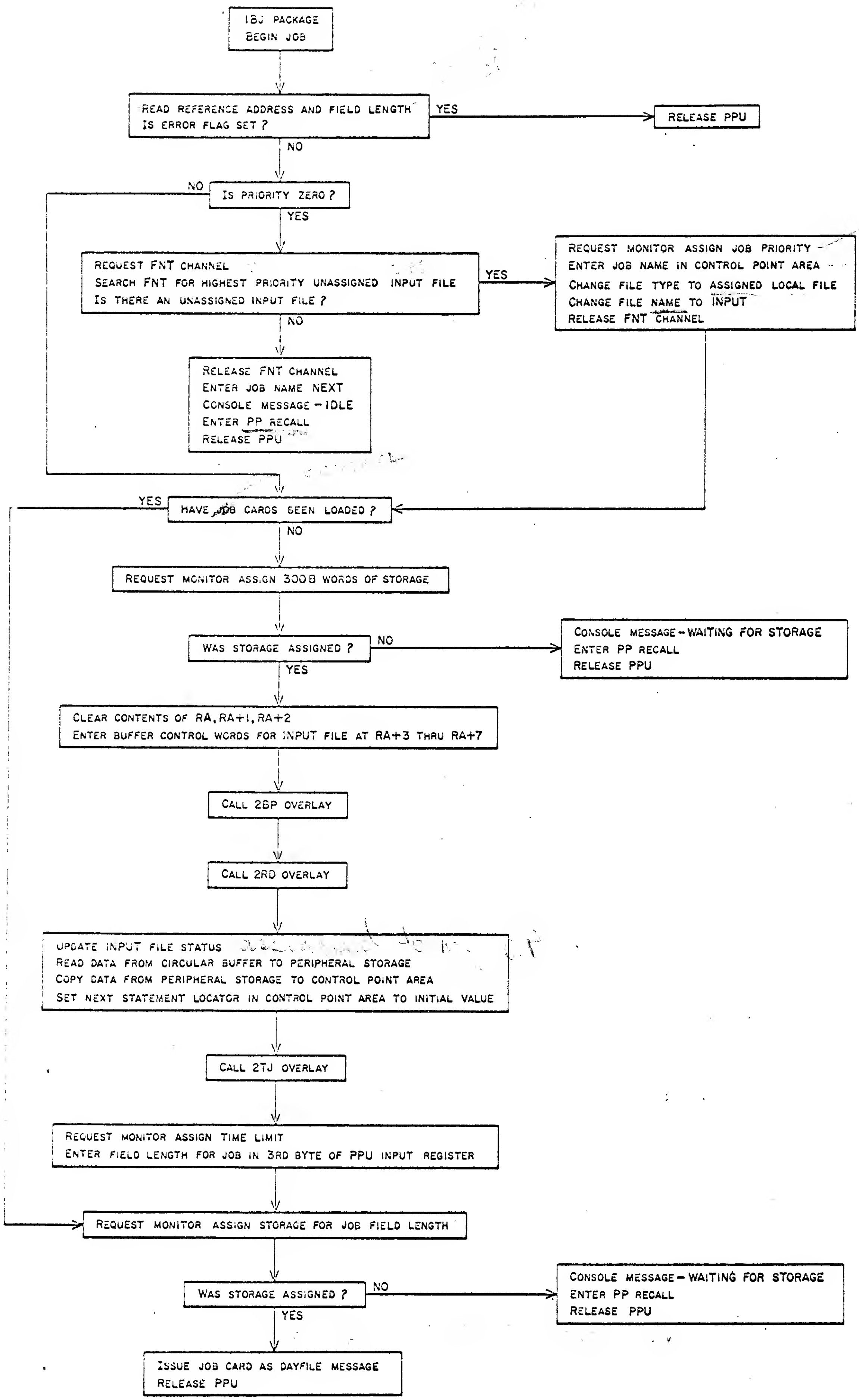
## 1BJ Routine Direct Core Parameters

| | | |
|---|---|---|
| 1000 | P75 | Address of Input register |
| 1100 | P50/54 | Contents of input register |
| | P74 | Address of control point |
| 1240 | (A) | Name of overlay to be loaded and executed |
| 1300 | P54 | Field Length (FL) from CP-20 |
| | P55 | Reference Address (RA) |
| | P57 | Address of INPUT FST entry. |
| | P63 | Lower 12 bits of IN = after control cards are read |
| | P65 | Lower 12 bits of OUT = |
| | P70 | 0001 (constant) |
| | P74 | Address of Control point |
| 1400 | (A) | Field length (in hundreds) needed |
| | P56 | Field length (FL) from CP-20. |
| | P74 | Address of control point |
| 1440 | P36 | Time Limit (TL) from JOB card (in tens) |
| | P37 | Field Length (FL) from JOB card (in hundreds) |
| | P55 | Reference Address (RA) |
| 1500 | P74 | Address of control point |




- FL, TIME, AND PRIORITY SET BY MTR AS REQUESTED BY 1BJ
- MTR MAY ALSO ADJUST RA

*JOB LOADING: "NEXT"*


1BJ PACKAGE
BEGIN JOB
READ REFERENCE ADDRESS AND FIELD LENGTH
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
IS PRIORITY ZERO ?
NO
YES
REQUEST FNT CHANNEL
SEARCH FNT FOR HIGHEST PRIORITY UNASSIGNED INPUT FILE
IS THERE AN UNASSIGNED INPUT FILE ?
YES
REQUEST MONITOR ASSIGN JOB PRIORITY
ENTER JOB NAME IN CONTROL POINT AREA
CHANGE FILE TYPE TO ASSIGNED LOCAL FILE
CHANGE FILE NAME TO INPUT
RELEASE FNT CHANNEL
NO
RELEASE FNT CHANNEL
ENTER JOB NAME NEXT
CONSOLE MESSAGE – IDLE
ENTER PP RECALL
RELEASE PPU
YES
HAVE JOB CARDS BEEN LOADED ?
NO
REQUEST MONITOR ASSIGN 300B WORDS OF STORAGE
WAS STORAGE ASSIGNED ?
NO
CONSOLE MESSAGE – WAITING FOR STORAGE
ENTER PP RECALL
RELEASE PPU
YES
CLEAR CONTENTS OF RA, RA+1, RA+2
ENTER BUFFER CONTROL WORDS FOR INPUT FILE AT RA+3 THRU RA+7
CALL 2BP OVERLAY
CALL 2RD OVERLAY
UPDATE INPUT FILE STATUS
READ DATA FROM CIRCULAR BUFFER TO PERIPHERAL STORAGE
COPY DATA FROM PERIPHERAL STORAGE TO CONTROL POINT AREA
SET NEXT STATEMENT LOCATOR IN CONTROL POINT AREA TO INITIAL VALUE
CALL 2TJ OVERLAY
REQUEST MONITOR ASSIGN TIME LIMIT
ENTER FIELD LENGTH FOR JOB IN 3RD BYTE OF PPU INPUT REGISTER
REQUEST MONITOR ASSIGN STORAGE FOR JOB FIELD LENGTH
WAS STORAGE ASSIGNED ?
NO
CONSOLE MESSAGE – WAITING FOR STORAGE
ENTER PP RECALL
RELEASE PPU
YES
ISSUE JOB CARD AS DAYFILE MESSAGE
RELEASE PPU

ROUTINE: 1DJ - Phase 3 print

PURPOSE: To monitor the processing of an OUTPUT file.

GENERAL: DSD calls 1DJ to a control point to print a jobs' output. The package appears as "PRINT" and is loaded at dead start when "AUTO" is typed or whenever "X.PRINT" is typed. It remains in recall state and is available to print an OUTPUT file when one is released.

METHOD:

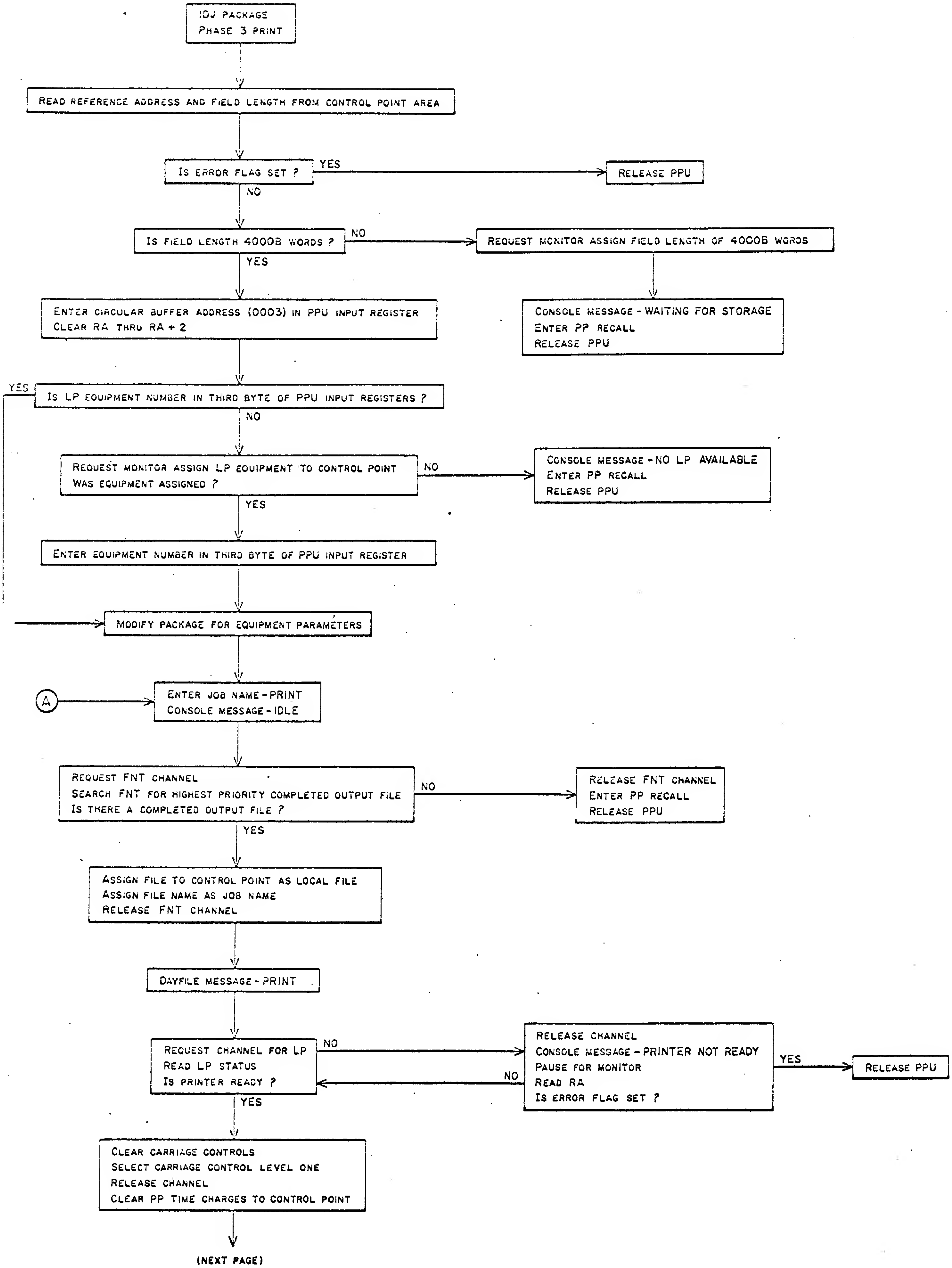
1. $4000_8$ words are requested from MTR. When memory has been allocated, a line printer is requested and the package is modified for the equipment parameters.

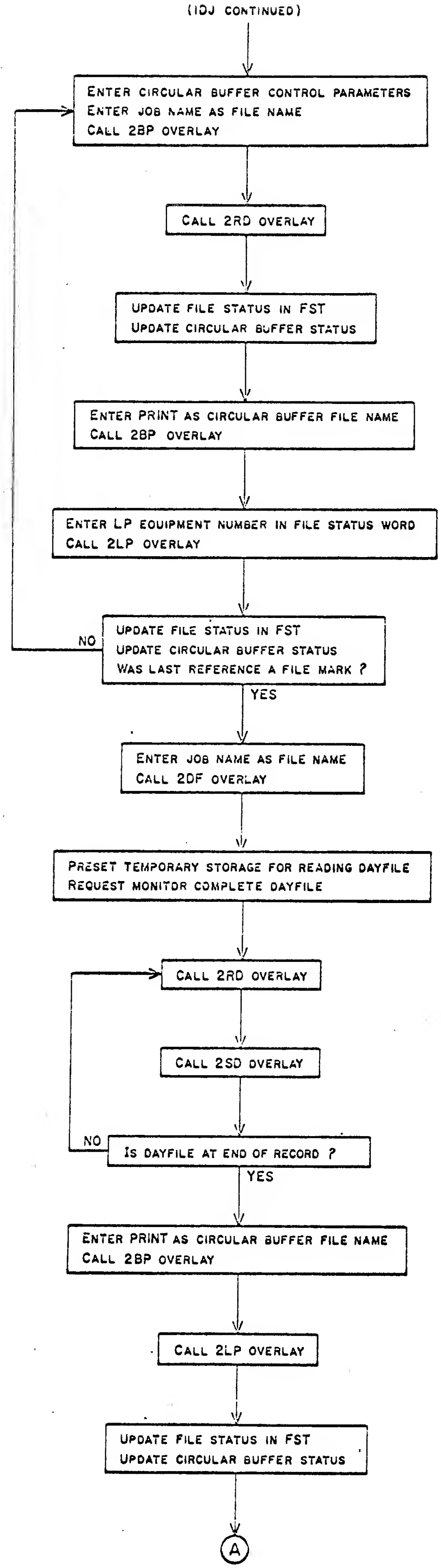
2. The FNT is searched for an "OUTPUT" file and the message "IDLE" is displayed until such a file is found. When found, a "PRINT" entry is made in the dayfile and when the printer becomes ready, the file name is changed to the job name in FNT. At the control point the job name appears instead of "PRINT" and the console message is changed from "IDLE" to "PRINT".

3. 2RD is called to read from the disk to the circular buffer in central memory. The reading continues until the end of the file is encountered or until the central memory buffer will not hold another full sector.

4. 2LP is then called to print this information and will continue printing until there is no more data in the buffer to print.

5. If an end-of-file has not yet been detected, control continues at step 3. When it is detected, the dayfile is searched for entries belonging to this job and then the entries are printed. Control reverts back to step 2.


IDJ PACKAGE
PHASE 3 PRINT
READ REFERENCE ADDRESS AND FIELD LENGTH FROM CONTROL POINT AREA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
IS FIELD LENGTH 4000B WORDS ?
NO
REQUEST MONITOR ASSIGN FIELD LENGTH OF 4000B WORDS
YES
ENTER CIRCULAR BUFFER ADDRESS (0003) IN PPU INPUT REGISTER
CLEAR RA THRU RA + 2
CONSOLE MESSAGE - WAITING FOR STORAGE
ENTER PP RECALL
RELEASE PPU
YES
IS LP EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTERS ?
NO
REQUEST MONITOR ASSIGN LP EQUIPMENT TO CONTROL POINT
WAS EQUIPMENT ASSIGNED ?
NO
CONSOLE MESSAGE - NO LP AVAILABLE
ENTER PP RECALL
RELEASE PPU
YES
ENTER EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTER
MODIFY PACKAGE FOR EQUIPMENT PARAMETERS
A
ENTER JOB NAME - PRINT
CONSOLE MESSAGE - IDLE
REQUEST FNT CHANNEL
SEARCH FNT FOR HIGHEST PRIORITY COMPLETED OUTPUT FILE
IS THERE A COMPLETED OUTPUT FILE ?
NO
RELEASE FNT CHANNEL
ENTER PP RECALL
RELEASE PPU
YES
ASSIGN FILE TO CONTROL POINT AS LOCAL FILE
ASSIGN FILE NAME AS JOB NAME
RELEASE FNT CHANNEL
DAYFILE MESSAGE - PRINT
REQUEST CHANNEL FOR LP
READ LP STATUS
IS PRINTER READY ?
NO
RELEASE CHANNEL
CONSOLE MESSAGE - PRINTER NOT READY
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
YES
CLEAR CARRIAGE CONTROLS
SELECT CARRIAGE CONTROL LEVEL ONE
RELEASE CHANNEL
CLEAR PP TIME CHARGES TO CONTROL POINT
(NEXT PAGE)


(IDJ CONTINUED)
ENTER CIRCULAR BUFFER CONTROL PARAMETERS
ENTER JOB NAME AS FILE NAME
CALL 2BP OVERLAY
CALL 2RD OVERLAY
UPDATE FILE STATUS IN FST
UPDATE CIRCULAR BUFFER STATUS
ENTER PRINT AS CIRCULAR BUFFER FILE NAME
CALL 2BP OVERLAY
ENTER LP EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2LP OVERLAY
NO
UPDATE FILE STATUS IN FST
UPDATE CIRCULAR BUFFER STATUS
WAS LAST REFERENCE A FILE MARK ?
YES
ENTER JOB NAME AS FILE NAME
CALL 2DF OVERLAY
PRESET TEMPORARY STORAGE FOR READING DAYFILE
REQUEST MONITOR COMPLETE DAYFILE
CALL 2RD OVERLAY
CALL 2SD OVERLAY
NO
IS DAYFILE AT END OF RECORD ?
YES
ENTER PRINT AS CIRCULAR BUFFER FILE NAME
CALL 2BP OVERLAY
CALL 2LP OVERLAY
UPDATE FILE STATUS IN FST
UPDATE CIRCULAR BUFFER STATUS
A

ROUTINE: 1LJ -- Phase One Card Load

PURPOSE: To build up an input file from the card reader onto the disk.

GENERAL: 1LJ is the "READ" package which is called in by DSD when "AUTO" is typed at dead start. When "READ" is assigned a control point, it remains in recall state and is available to read a job whenever the card reader becomes ready.

METHOD:

1. The job name READ is stored in CP(21). The error flag is checked and if an error is sensed, the PP is released. READ must be reassigned when it is needed again.

2. If $4000_8$ words (FL) have not been assigned, the routine requests the storage and puts itself into PP recall.

3. A circular buffer address (0003) is entered into the PP input register and the first 3 words (RA—RA+2) are cleared. Any central program must have 3 words reserved for system communication so that means the circular buffer parameters are located at RA+3.

4. FIRST = IN = OUT = $10_8$ are the preset buffer parameters and LIMIT = $4000_8$.

5. Upon entry the third byte of the input register may contain the equipment number of the card reader. If it does not, then MTR is asked for the assignment. The number will come back in the first byte of the message buffer and then is transferred to the third byte of the input register.

6. If the assignment was not completed, "NO CR AVAILABLE" is stored in CP(30) and the PP put into recall.

7. The above 6 steps are initialization procedures and are not repeated unless "READ" is dropped and must be reassigned.

8. "READ" appears as the job name in the CP and "IDLE" as a console message when no reading is being done.

9. The channel from the card reader entry in the equipment status table (EST) is requested and then the status of the card reader is checked. If the reader is not ready, the PP is put into recall and released.

10. After the card reader is found to be ready, the file name "READ" and a buffer status of $10_8$ meaning requested coded read is entered into BA.

11. 2BP is called to check the legality of the buffer parameters.

12. The equipment number of the card reader from the input register is stored in the FST entry.

13. 2RC is then called to read one card.

14. The FST entry is updated and stored as is the buffer status word (BA). Both reflect an $11_8$ condition completed coded read.

15. 2TJ is called to translate the job card. The job name is entered in PP(30) from 2TJ and is transferred to CP(21). Therefore, the control point assigned to READ has a new job name (from job card) and a console message of "READ" instead of "IDLE". A dayfile entry of the job name and READ is made.

16. Next READ in BA is replaced with the job name and the buffer status is changed to request coded write ($14_8$).

17. Again 2BP is called to verify the buffer parameters. Every write operation on the disk is terminated with an EOF record so that if a file mark was requested it is not completed so that two file marks will not be written.

18. 2WD is called to write the contents of the buffer of the disk.

19. Upon reentry to 1LJ, the FST entry and the buffer status (BA) is updated to reflect a completed coded write.

20. The file name READ and buffer status of $10_8$ - requested coded read - is again entered into BA.

21. 2BP is called to determine the legality of the buffer parameters and the card reader equipment number is placed in the third byte of the input register for 2RC.

22. After 2RC returns control to 1LJ, the FST and buffer status are updated to reflect a completed coded read.

23. If a file mark was not read, then the job has not been completely read in. The contents of the buffer are written on the disk and more cards read until a 6-7-8-9 card is found.

24. When a file separator card is sensed, an MTR request (04) to update the PP running time at the control point for the requesting processor is issued. The time is converted to decimal and sent to the dayfile in the form PPXXXX sec.

25. In order to release the job to the system the job name is stored in BA and 2BP is called for a final check of the buffer parameters. The disk file is rewound by setting the current track to the beginning track in the

FST. Also the current sector byte is cleared and the last buffer status is set to 01. The priority is added to the FNT entry and the control point assignment byte is cleared. Therefore, the input file is released and ready for MTR to assign it a control point for execution.

26. "READ" with a $10_8$ request is again entered into BA of the circular buffer and 2BP is called to check the parameters. An FST entry is cleared in preparation for a new file and a check is made for a ready card reader.

27. If a card reader is not ready, the PP is put into recall so that it will be able to detect when the card reader becomes ready.

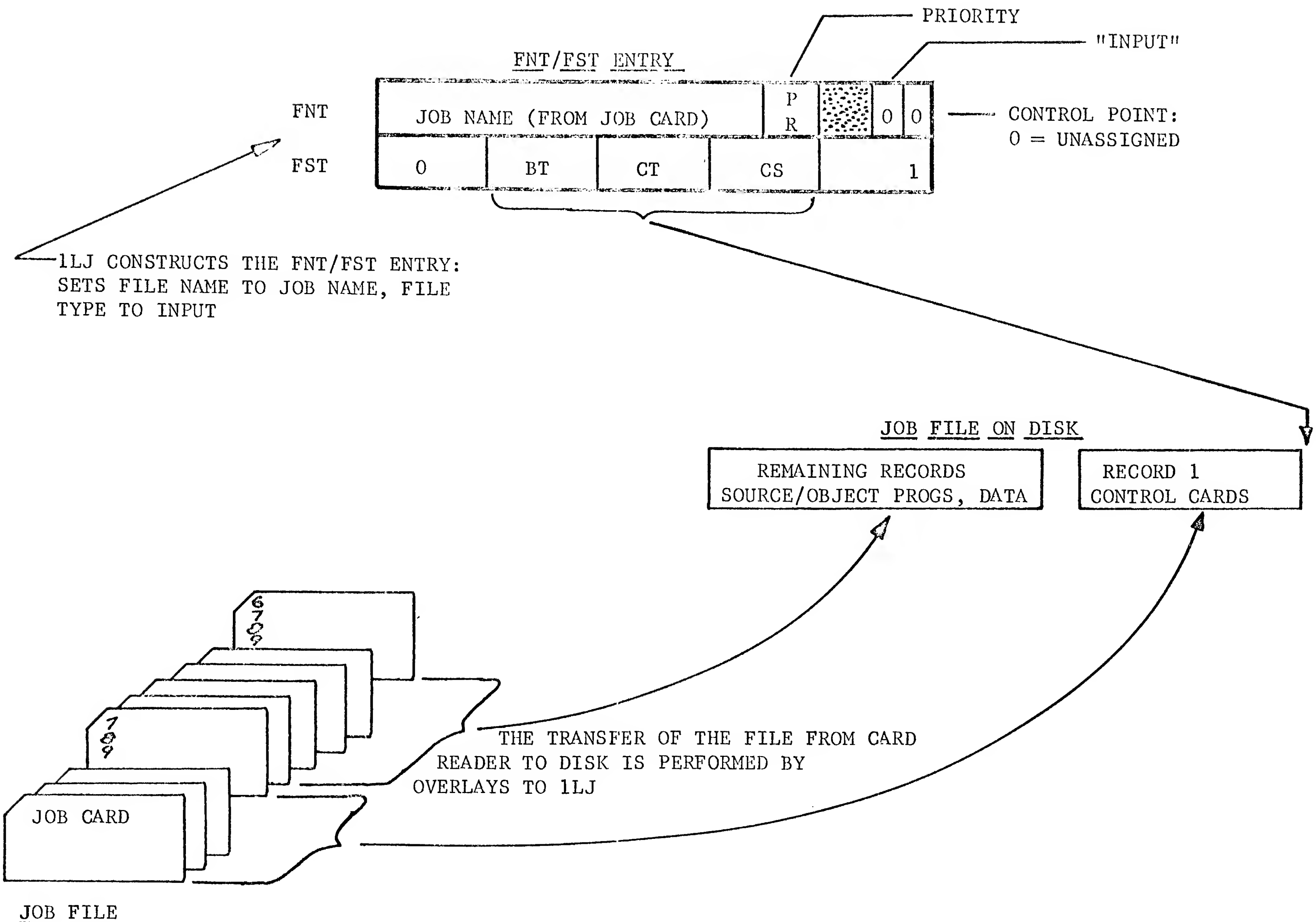
## 1LJ Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1500, 1440, 1600, 1040 |
| 1040 | Process Job | 1700, 1400, 530, 1100, 1300, 1200 |
| 1100 | Dump Buffer | 1400 |
| 1200 | Release Job | 1400 |
| 1300 | Record Time | 4-760, 530 |
| 1400 | Call RPL Package | 2000 |
| 1440 | Request CR | 22-760, 12-760, 100 |
| 1500 | Enter CP Status | 10-760, 12-760, 100, 1740 |
| 1600 | Sense CR Ready | 740, 750, 12-760, 100 |
| 1700 | Load Buffer | 1400 |
| 1740 | Preset Buffer Parameters | |

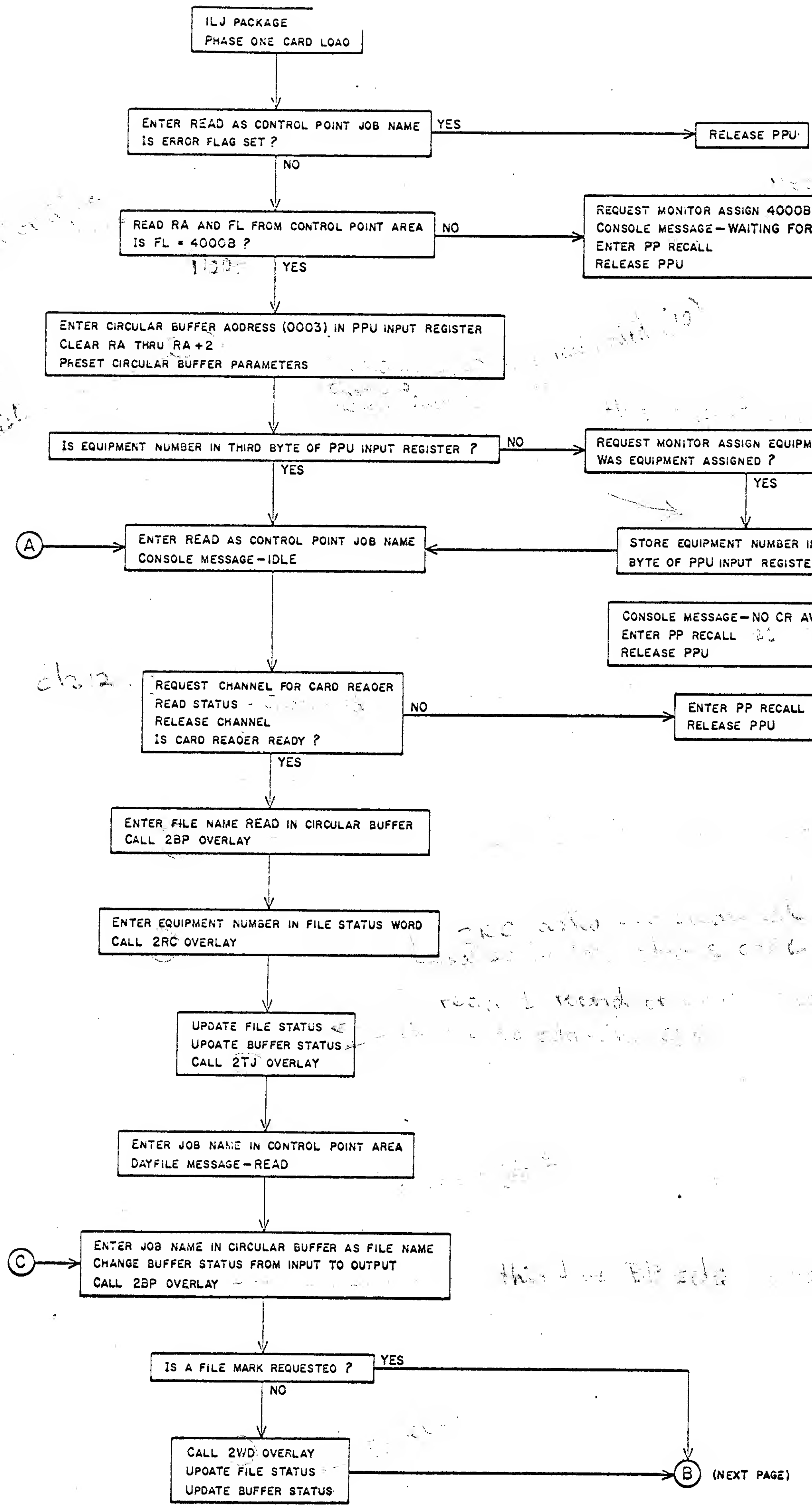
## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P50-54 | Input register |
| | P70 | Constant 1 |
| | P71 | Constant 180 |
| | P72 | Constant 1000 |
| | P75 | Input register address |
| 1040 | P10/14 | Zero word |
| | P20/24 | FST entry |
| | P74 | CP address |
| 1100 | P10/14 | CP(21) |
| | P20/24 | FST entry |
| | P40/44 | File control word (BA) |
| | P50/54 | Input register |
| | P55 | RA |
| | P57 | FST address |
| 1200 | P10/14 | CP(21), zero word |
| | P20/24 | FST entry |
| | P35 | Job priority |
| | P55 | RA |
| | P57 | FST address |
| | P74 | CP address |

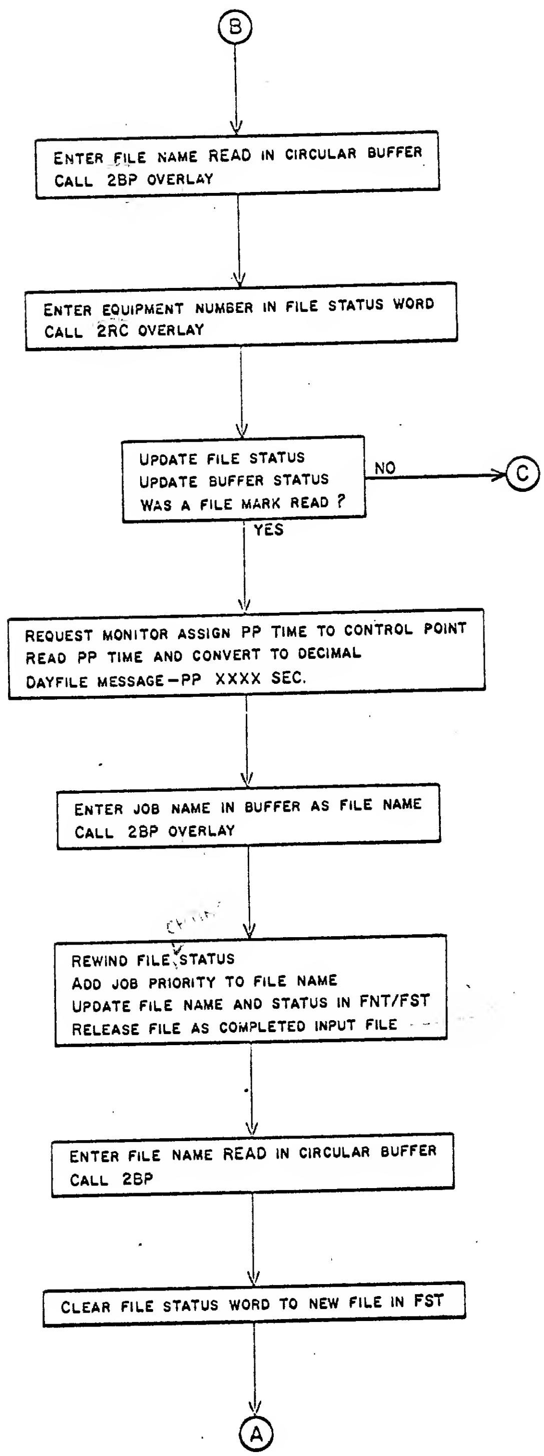
| | | |
|---|---|---|
| 1300 | P10/14 | PP time - CP(24) |
| | P74 | CP address |
| 1400 | (A) | Package name |
| | P01 | RPL ordinal |
| | P02/03 | Package name |
| | P10/14 | RPL entry |
| 1440 | P01 | Constant 2 |
| | P10/14 | Message buffer |
| | P50/54 | Input register |
| | P74 | CP address |
| | P77 | Message buffer address |
| 1500 | P01 | Constant 3 |
| | P10/14 | CP(20), zero word |
| | P54 | Constant 3 |
| | P50/54 | Input register |
| | P55 | RA |
| | P56 | FL |
| | P74 | CP address |
| 1600 | P01 | CR status |
| | P10/14 | EST status |
| | P20/24 | EST entry |
| | P50/54 | Input register |
| | P74 | CP address |
| 1700 | P20/24 | FST entry |
| | P40/44 | File control word (BA) |
| | P50/54 | Input register |
| | P55 | RA |
| | P57 | FST address |
| 1740 | P10/14 | Zero |
| | P14 | $10_8$ |
| | P54 | Constant 3 |
| | P55 | RA |
| | P56 | FL |




JOB LOADING: "READ"


ILJ PACKAGE
PHASE ONE CARD LOAD
ENTER READ AS CONTROL POINT JOB NAME
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
READ RA AND FL FROM CONTROL POINT AREA
IS FL = 4000B ?
NO
REQUEST MONITOR ASSIGN 4000B FIELD LENGTH
CONSOLE MESSAGE—WAITING FOR STORAGE
ENTER PP RECALL
RELEASE PPU
YES
ENTER CIRCULAR BUFFER ADDRESS (0003) IN PPU INPUT REGISTER
CLEAR RA THRU RA+2
PRESET CIRCULAR BUFFER PARAMETERS
IS EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTER ?
NO
REQUEST MONITOR ASSIGN EQUIPMENT TYPE CR
WAS EQUIPMENT ASSIGNED ?
NO
YES
YES
STORE EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTER
A
ENTER READ AS CONTROL POINT JOB NAME
CONSOLE MESSAGE—IDLE
CONSOLE MESSAGE—NO CR AVAILABLE
ENTER PP RECALL
RELEASE PPU
REQUEST CHANNEL FOR CARD READER
READ STATUS
RELEASE CHANNEL
IS CARD READER READY ?
NO
ENTER PP RECALL
RELEASE PPU
YES
ENTER FILE NAME READ IN CIRCULAR BUFFER
CALL 2BP OVERLAY
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2RC OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
CALL 2TJ OVERLAY
ENTER JOB NAME IN CONTROL POINT AREA
DAYFILE MESSAGE—READ
C
ENTER JOB NAME IN CIRCULAR BUFFER AS FILE NAME
CHANGE BUFFER STATUS FROM INPUT TO OUTPUT
CALL 2BP OVERLAY
IS A FILE MARK REQUESTED ?
YES
NO
CALL 2WD OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
B
(NEXT PAGE)

(ILJ CONTINUED)



ROUTINE: 1LT Phase One Tape Load

PURPOSE: To load jobs from a magnetic tape onto the disk until an empty file is encountered.

GENERAL: The package is called by DSD after the operator types "X.LOAD." at the console. The control point for the package is specified in the input register.

METHOD:

1. Initialization of the routine involves the following steps:

   a. If the requested control point has a job name, the package is released.

   b. Otherwise, the job name LOAD is set in the CP(21) for display purposes.

   c. $10000_8$ words of central memory are requested to be used as a buffer for reading tape and writing disk.

   d. If MTR does not assign $10000_8$ words, the package is released.

   e. Otherwise, the CM buffer is set up as follows:

| | | | |
|---|---|---|---|
| RA = RA+1 = RA+2 = | 0 | | |
| RA+3 = | 0 | | File Name and Buffer Status |
| RA+4 = RA+5 = RA+6 = | 00 | 04 | FIRST, IN, OUT |
| RA+7 = | 00 | 010000 | LIMIT |

   f. The buffer address, 0003, is stored in the PPU input register (internal) for future reference by the package.

   g. A tape assignment is requested of the operator by storing REQUEST TAPE in CP(30-37).

   h. A function 17 request is sent to MTR while waiting for the operator to assign the tape. This function is repeated until the tape is assigned. The equipment number specified by the operator is contained in CP(22).

   i. The equipment number is stored in PPU input register (internal) for future reference by the package.

2. The following steps occur for the initialization of each file (job):

   a. RA+3 is (re) set as follows:

TAPE......10 in order to read the tape files.

b. 2BP is called to verify the buffer parameters and to set up direct core parameters for 2RT.

c. 2RT is called to read information from the tape and store it in buffer in central memory.

d. The file status (LBS field) in FST is updated (odd value) to reflect the record(s) just read.

e. The buffer status (at RA) is updated (odd value) to reflect the record(s) just read.

f. If a file mark was read at this point, it would have been the second consecutive file mark and, therefore, the package (1LT) is released.

g. Otherwise, 2TJ is called to set up the job name and priority in direct core cells.

h. The job name is in the CP for display and dayfile accounting purposes.

i. The message LOAD is sent to the dayfile.

3. The following steps occur as a loop for loading the tape records onto the disk:

   a. The job name (from CP) is stored as file name before writing disk so that FNT contains the job name of type input.

   b. 2BP is called to set up direct core parameters for 2WD, i.e., also assigns the new file.

   c. 2WD is called to write the bufferin central memory onto the disk, if a file mark was not requested. The file marks are automatically handled by 2WD on every write.

   d. Again, the FST word and the buffer status are updated to reflect the record(s) just written.

   e. The buffer is again loaded as specified before in steps 2) a., b., c., d., e.

4. When a file mark is encountered on the tape (and the record(s) are written on disk), the following steps are performed to release the disk file (job) just written.

   a. The job name is stored as the file name in order to call 2BP to set up direct core parameters for rewinding the file.

   b. The file (on disk) is rewound by making the following changes to FST.

i. setting current track=beginning track

ii. setting current sector=0

iii. setting last buffer status=0001

c. The priority, from the job card, is entered into FNT.

d. The file type is set to input.

e. The file status is cleared from file TAPE by a call to 2BP and resetting FST.

NOTES: PPU time used to load the jobs on the disk is not charged to the individual jobs.

The package 1LT is released without completing the tape to disk operation if any of the following conditions arise:

1. too many control cards in a job.

2. illegal parameters on the job card.

3. no tracks are available on disk.

4. the track limit (512 tracks) is exceeded for a job.

5. the operator drops the CP.

When the package (1LT) is released, either normally or prematurely, the files (tape and disk-FNT/FST), equipment (EST), and storage (CP(20)) are released by a special section of 1AJ. This section releases these items for control points not using the CPU but using CM for buffers. 1AJ detects this when a CP has a zero (0) priority. 1AJ is entered to release the package by the master loop in MTR.

The dayfile message LOAD is written via MTR function 01 and resident routine located at $530_8$.

Since the package is immediately released if $10000_8$ words are not available from MTR, the operator should call LOAD after dead start. Otherwise, he will have to wait for the CP's to be relatively inactive in order not to run into any storage conflicts.

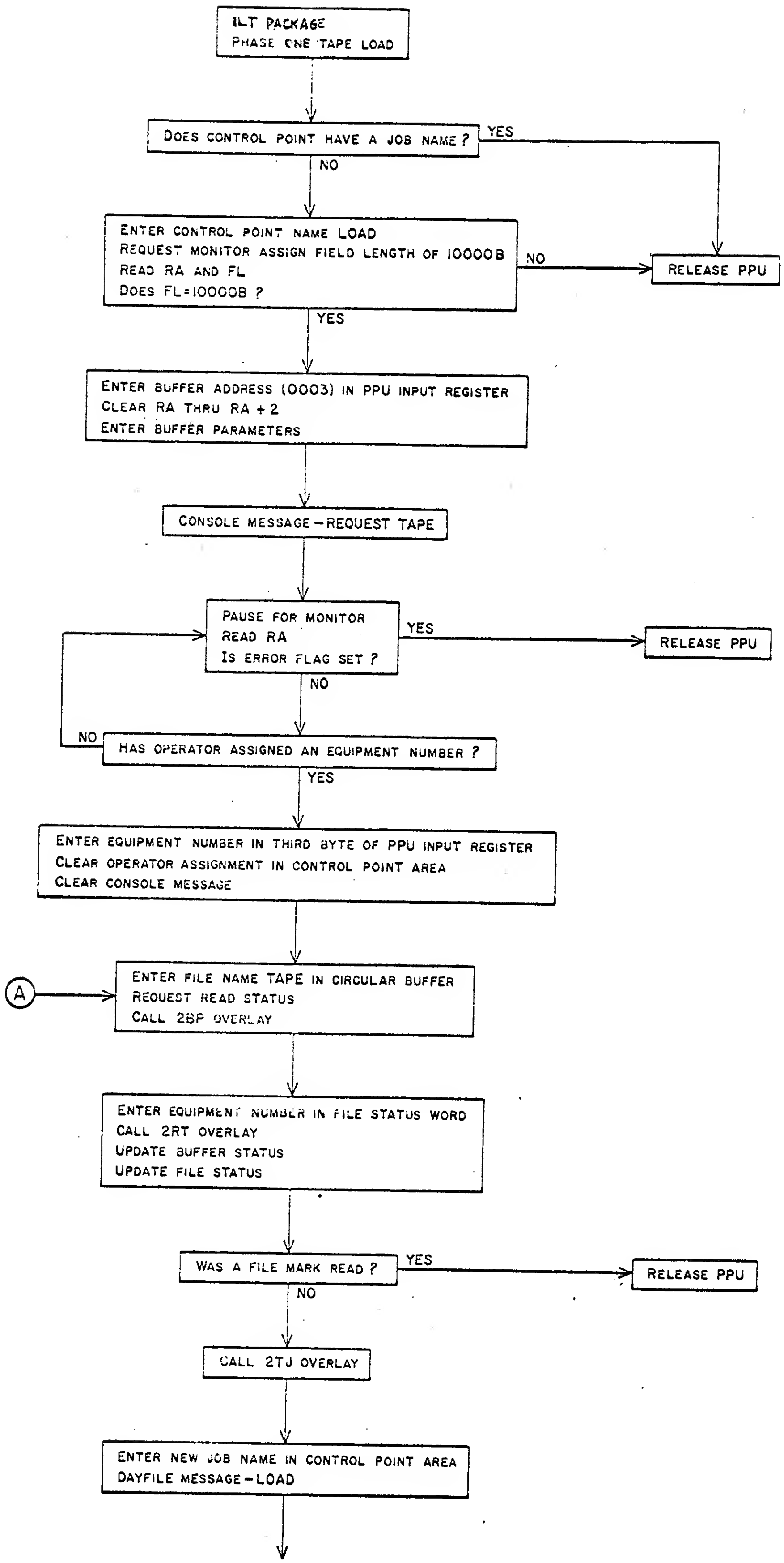
All overlays called by 1LT must be in RPL since PLD is <u>not</u> searched when calling the overlays. These overlays include 2BP, 2RT, 2WD, 2TJ.

## 1LT Load Tape Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1300, 1440, 1240, 12-760, 1400, 530, 1100, 1160 |
| 1100 | Dump Buffer | 1400 |
| 1160 | Release Job | 1400 |
| 1240 | Load Buffer | 1400 |
| 1300 | Enter CP Status | 10-760, 12-760 |
| 1400 | Call RPL Package | |
| 1400 | Request Tape | 17-760, 12-760 |

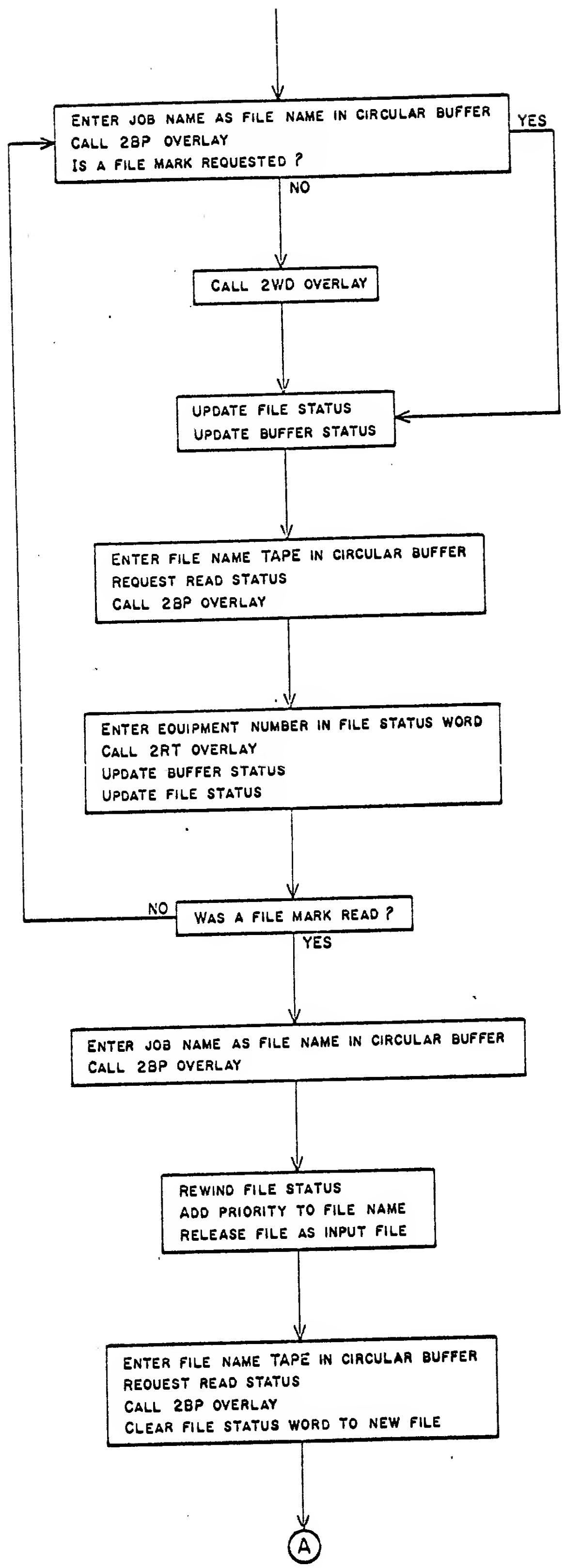
## Direct Core Cells

| | |
|---|---|
| P20/24 | FST entry for file sent to 2BP |
| P30/34 | Job name from job card set up by 2TJ |
| P35 | Priority from job card set up by 2TJ |
| P40/44 | File Control Word+Buffer Status (same as RA+3) |
| P50/54 | Input Register |
| P55 | RA from CP(20) |
| P56 | FL from CP(20) |
| P57 | FST entry address set up by 2BP |
| P70 | 0001 (constant) |
| P71 | 0100 (constant) |
| P72 | 1000 (constant) |
| P74 | Control point address |
| P75 | Input register address |


ILT PACKAGE
PHASE ONE TAPE LOAD
DOES CONTROL POINT HAVE A JOB NAME ?
YES
NO
ENTER CONTROL POINT NAME LOAD
REQUEST MONITOR ASSIGN FIELD LENGTH OF 10000B
READ RA AND FL
DOES FL=10000B ?
NO
RELEASE PPU
YES
ENTER BUFFER ADDRESS (0003) IN PPU INPUT REGISTER
CLEAR RA THRU RA + 2
ENTER BUFFER PARAMETERS
CONSOLE MESSAGE—REQUEST TAPE
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
HAS OPERATOR ASSIGNED AN EQUIPMENT NUMBER ?
YES
ENTER EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTER
CLEAR OPERATOR ASSIGNMENT IN CONTROL POINT AREA
CLEAR CONSOLE MESSAGE
A
ENTER FILE NAME TAPE IN CIRCULAR BUFFER
REQUEST READ STATUS
CALL 2BP OVERLAY
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2RT OVERLAY
UPDATE BUFFER STATUS
UPDATE FILE STATUS
WAS A FILE MARK READ ?
YES
RELEASE PPU
NO
CALL 2TJ OVERLAY
ENTER NEW JOB NAME IN CONTROL POINT AREA
DAYFILE MESSAGE—LOAD

(NEXT PAGE)

(ILT CONTINUED)



ROUTINE: 1TD - Phase 3 Tape Dump

PURPOSE: To dump completed output files on tape in order of priority for off-line printing.

GENERAL: 1TD is assigned a PP and a control point when "X.DUMP." is typed. Whenever all output files are dumped, the package is released.

METHOD:

1. "DUMP" is assigned as the job name for the control point. $10000_8$ words of central memory are required for the buffer and if it is not assigned, the PP is released.

2. The message "REQUEST TAPE" appears as the third line of the control point. The operator must enter "X.ASSIGN YY.", where YY is a tape equipment number.

3. The FNT is searched for the highest priority output file.

4. The file is assigned to the control point as a local file and the job name from FNT is set into CP(21). The job name replaces "DUMP" at the control point and "DUMP" is displayed as the console message.

5. The central memory buffer is filled by 2RD.

6. 2WT is called and the tape equipment number is set in FST. If the tape assigned is ½", a return jump is made to the BCD write coding in 2WT. A 1" tape assignment giver the binary write of 2WT control.

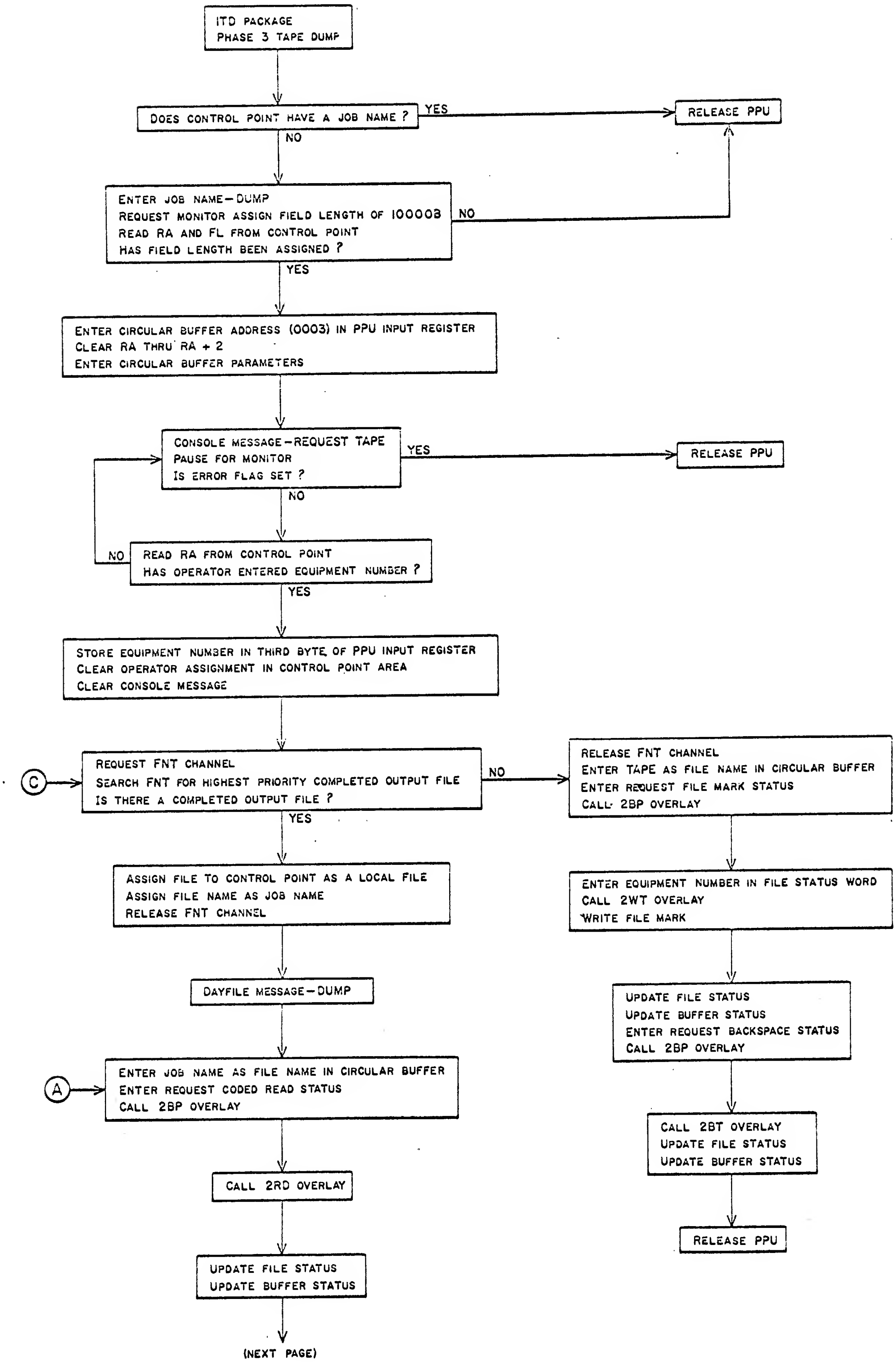
7. When the buffer is emptied the FST and buffer status are updated. No file mark is written between jobs.

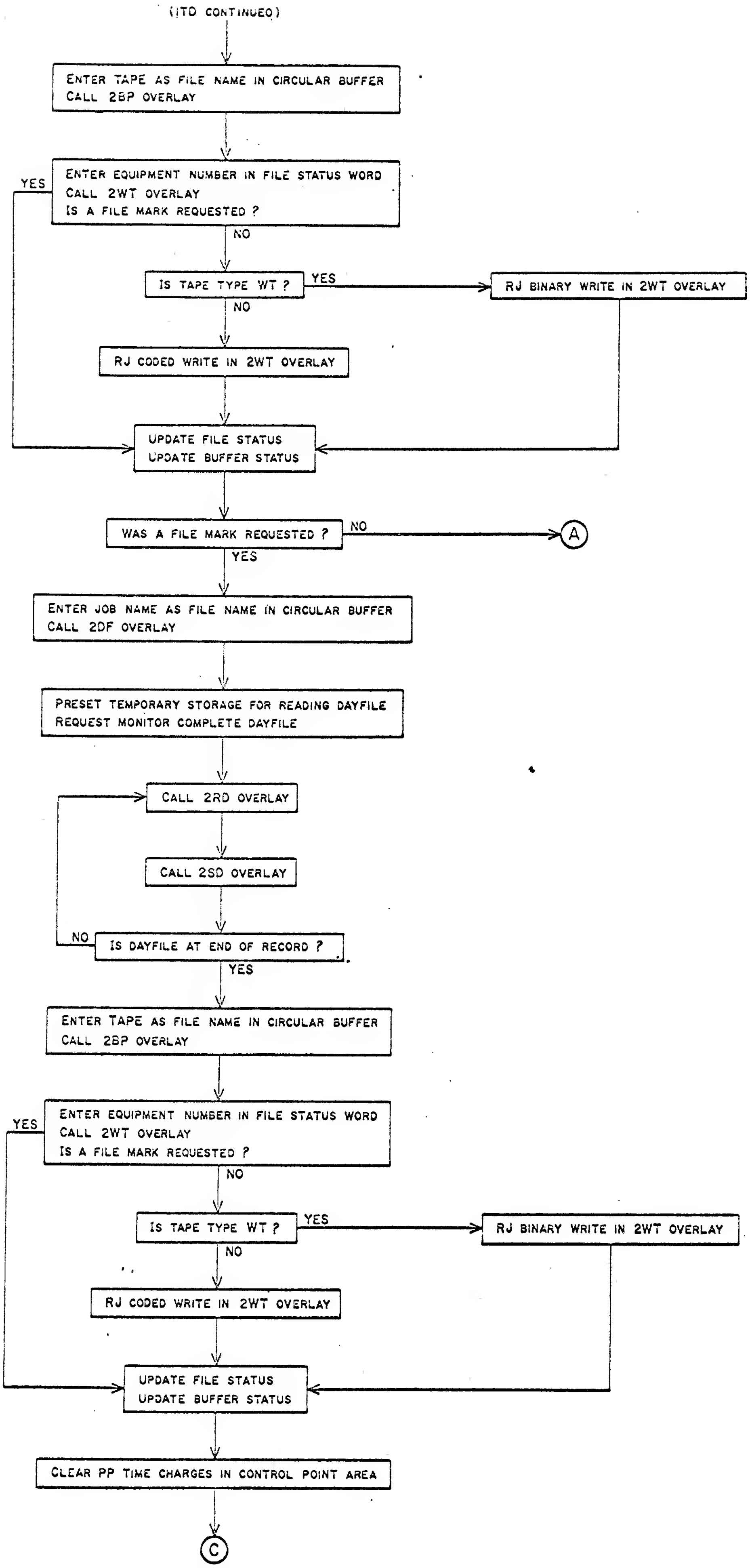
8. Whenever the job output file has been dumped and a file mark requested, 2DF is called to drop the disk tracks used by the file.

9. 2RD and 2SD search the dayfile for entries pertaining to the job and they are written after the job output by 2WT.

10. All PP time charges at the control point are cleared.

11. Again FNT is searched for the highest priority output file. When no more output files exist, a file mark is written and then the tape is backspaced over it. The tape is left in this position so that more dumps may be added.


ITD PACKAGE
PHASE 3 TAPE DUMP
DOES CONTROL POINT HAVE A JOB NAME ?
YES
RELEASE PPU
NO
ENTER JOB NAME—DUMP
REQUEST MONITOR ASSIGN FIELD LENGTH OF 10000B
READ RA AND FL FROM CONTROL POINT
HAS FIELD LENGTH BEEN ASSIGNED ?
NO
YES
ENTER CIRCULAR BUFFER ADDRESS (0003) IN PPU INPUT REGISTER
CLEAR RA THRU RA + 2
ENTER CIRCULAR BUFFER PARAMETERS
CONSOLE MESSAGE—REQUEST TAPE
PAUSE FOR MONITOR
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
READ RA FROM CONTROL POINT
HAS OPERATOR ENTERED EQUIPMENT NUMBER ?
YES
STORE EQUIPMENT NUMBER IN THIRD BYTE OF PPU INPUT REGISTER
CLEAR OPERATOR ASSIGNMENT IN CONTROL POINT AREA
CLEAR CONSOLE MESSAGE
C
REQUEST FNT CHANNEL
SEARCH FNT FOR HIGHEST PRIORITY COMPLETED OUTPUT FILE
IS THERE A COMPLETED OUTPUT FILE ?
NO
RELEASE FNT CHANNEL
ENTER TAPE AS FILE NAME IN CIRCULAR BUFFER
ENTER REQUEST FILE MARK STATUS
CALL 2BP OVERLAY
YES
ASSIGN FILE TO CONTROL POINT AS A LOCAL FILE
ASSIGN FILE NAME AS JOB NAME
RELEASE FNT CHANNEL
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2WT OVERLAY
WRITE FILE MARK
DAYFILE MESSAGE—DUMP
UPDATE FILE STATUS
UPDATE BUFFER STATUS
ENTER REQUEST BACKSPACE STATUS
CALL 2BP OVERLAY
A
ENTER JOB NAME AS FILE NAME IN CIRCULAR BUFFER
ENTER REQUEST CODED READ STATUS
CALL 2BP OVERLAY
CALL 2BT OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
CALL 2RD OVERLAY
RELEASE PPU
UPDATE FILE STATUS
UPDATE BUFFER STATUS
(NEXT PAGE)


(ITD CONTINUED)
ENTER TAPE AS FILE NAME IN CIRCULAR BUFFER
CALL 2BP OVERLAY
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2WT OVERLAY
IS A FILE MARK REQUESTED ?
YES
NO
IS TAPE TYPE WT ?
YES
RJ BINARY WRITE IN 2WT OVERLAY
NO
RJ CODED WRITE IN 2WT OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
WAS A FILE MARK REQUESTED ?
NO
A
YES
ENTER JOB NAME AS FILE NAME IN CIRCULAR BUFFER
CALL 2DF OVERLAY
PRESET TEMPORARY STORAGE FOR READING DAYFILE
REQUEST MONITOR COMPLETE DAYFILE
CALL 2RD OVERLAY
CALL 2SD OVERLAY
NO
IS DAYFILE AT END OF RECORD ?
YES
ENTER TAPE AS FILE NAME IN CIRCULAR BUFFER
CALL 2BP OVERLAY
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
CALL 2WT OVERLAY
IS A FILE MARK REQUESTED ?
YES
NO
IS TAPE TYPE WT ?
YES
RJ BINARY WRITE IN 2WT OVERLAY
NO
RJ CODED WRITE IN 2WT OVERLAY
UPDATE FILE STATUS
UPDATE BUFFER STATUS
CLEAR PP TIME CHARGES IN CONTROL POINT AREA
C

PROGRAM: 2BP -- Read Buffer Parameter

PURPOSE: To examine the buffer arguments for correctness, enter file name in FNT, and reserve the file.

GENERAL: This routine is called by 1AJ, 1BJ, 1DJ, 1LJ, 1LT, 1TD, CIO to check the buffer arguments for range and validity. It also enters file name in the FNT, reserves the file if possible. The following error messages are produced: BUFFER ARG ERROR, and FNT LIMIT.

METHOD:

1. Read buffer status and arguments.
2. Move the arguments to a two word/entry table at P60.
3. Check for argument region out of field limit range. If in error, display in dayfile - BUFFER ARG ERROR, issue a FC of 13B (abort CP), and exit to PP monitor loop.
4. Check for LIMIT over field limit and go to the error procedure if it is.
5. Check for OUT > LIMIT.
6. Check for IN > LIMIT.
7. Check for OUT < FIRST.
8. Check for IN < FIRST.
9. Check each character of file name to first blank for less than 37. If an error is detected, go through same error procedure as above. Also senses inserted characters after the first blank as errors. Finally, it checks to make certain file name is non-blank.
10. Searches FNT for the file name and matching CP number. On a find, it saves FST entry address.
11. If the file was not found in FNT, it locks out other PP's from the FNT. A blank entry is found and the name is entered with its CP, file set as local, and priority of zero. A blank entry is written into FST. Channel 15 is released thereby allowing other PP's into FNT, and FST address is saved.
12. Request channel 14 (FST lock out channel). Check LBS field of FST for file reserved (even number - reserved). If it is not reserved, reserve it (set FST odd), release channel 14 and exit.
13. If it is reserved, release channel 14, and issue a 17B

function to allow the monitor to move central storage.

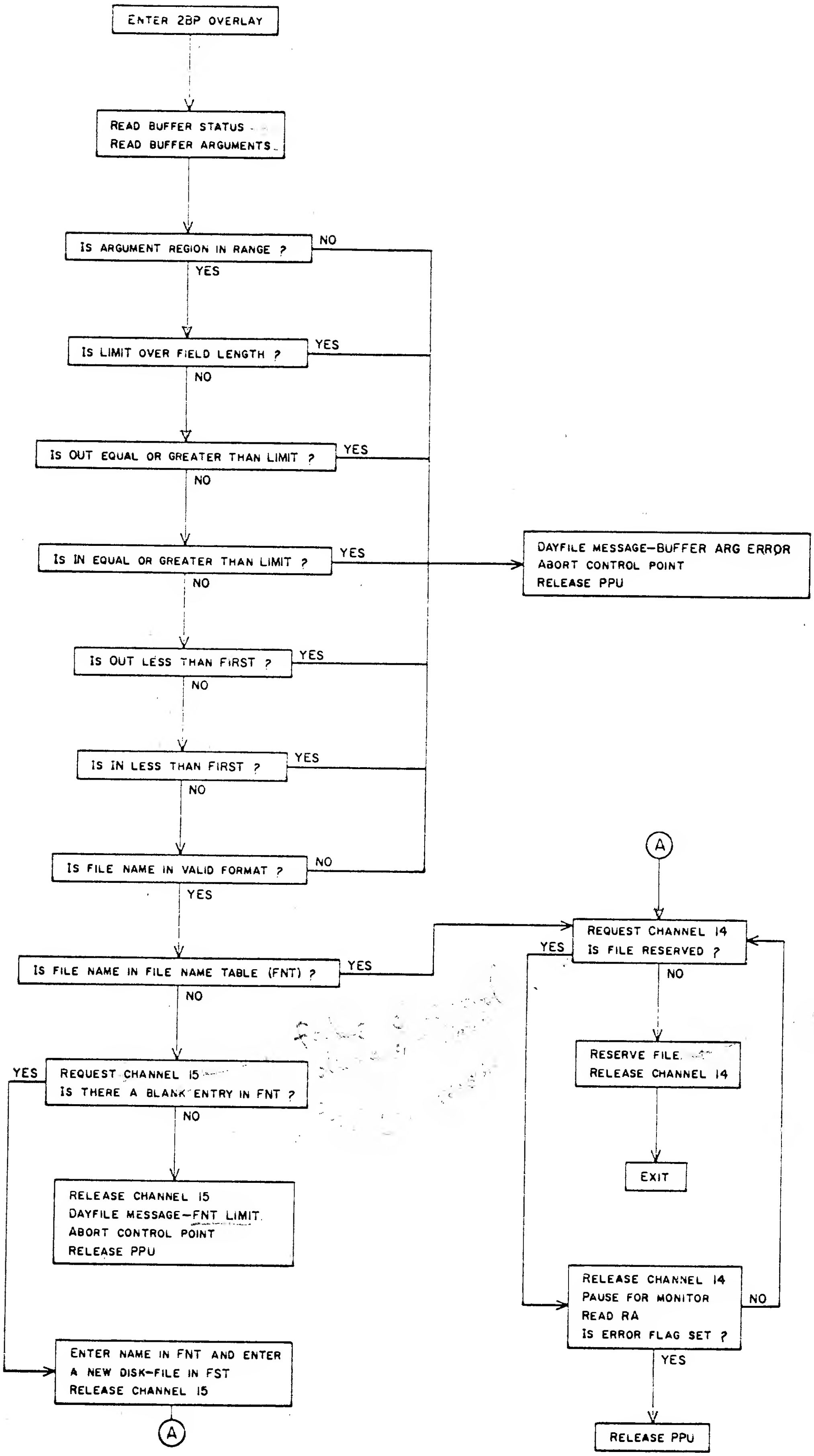
14. Read CP status. Save reference address. If error flag is not set, go back to No. 12 above and continue. If flag is set, release PP(12B) and exit to resident PP program.

## 2BP Routines

| | | |
|---|---|---|
| 2000 | Main Program | 2350, 2300, 2150, 2100 |
| 2100 | Alter File Status | 14-740, 14-750, 17-760, 12-760, 100 |
| 2150 | Search FNT | 15-740, 15-750, 530, 13-760, 100 |
| 2300 | Verify File Name | 530, 13-760, 100 |
| 2350 | Verify Argument Values | 530, 13-760, 100 |

## Direct Core Cells

| | | |
|---|---|---|
| 2000 | P50/54 | Input register (Buffer address) |
| | P40/44 | Status |
| | P01 | Counter for buffer parameters |
| | P02 | Address for storing buffer arguments |
| | P60/70 | Buffer arguments (FIRST, IN, OUT, LIMIT) |
| | P10/14 | Temporary storage for buffer arguments |
| 2100 | P57 | File status address |
| | P20/24 | File status |
| | P45 | Last buffer status from FST |
| | P40/44 | Buffer status |
| | P74 | Control Point address |
| | P10/14 | CP status |
| | P55 | Reference address |
| 2150 | P20/24 | FNT address and limit |
| | P10/14 | FNT entry |
| | P40/44 | Buffer status (Name of file) |
| | P51 | Input register (CP for file) |
| | P57 | File status address |
| 2300 | P01 | Address of file name |
| | P40/44 | Buffer status (file name) |
| 2350 | P53/54 | Argument address |
| | P56 | Field length |
| | P60/61 | FIRST |
| | P62/63 | IN |
| | P64/65 | OUT |
| | P66/67 | LIMIT |


ENTER 2BP OVERLAY
READ BUFFER STATUS
READ BUFFER ARGUMENTS
IS ARGUMENT REGION IN RANGE ?
NO
YES
IS LIMIT OVER FIELD LENGTH ?
YES
NO
IS OUT EQUAL OR GREATER THAN LIMIT ?
YES
NO
IS IN EQUAL OR GREATER THAN LIMIT ?
YES
NO
DAYFILE MESSAGE—BUFFER ARG ERROR
ABORT CONTROL POINT
RELEASE PPU
IS OUT LESS THAN FIRST ?
YES
NO
IS IN LESS THAN FIRST ?
YES
NO
IS FILE NAME IN VALID FORMAT ?
NO
YES
IS FILE NAME IN FILE NAME TABLE (FNT) ?
YES
NO
A
REQUEST CHANNEL 14
IS FILE RESERVED ?
YES
NO
RESERVE FILE
RELEASE CHANNEL 14
EXIT
REQUEST CHANNEL 15
IS THERE A BLANK ENTRY IN FNT ?
YES
NO
RELEASE CHANNEL 15
DAYFILE MESSAGE—FNT LIMIT
ABORT CONTROL POINT
RELEASE PPU
RELEASE CHANNEL 14
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
YES
ENTER NAME IN FNT AND ENTER
A NEW DISK-FILE IN FST
RELEASE CHANNEL 15
A
RELEASE PPU

ROUTINE: 2BT - BACKSPACE TAPE

PURPOSE: To backspace a block of small binary or BCD data on tape and set buffer addresses accordingly.

GENERAL: The 2BT routine is called in once the backspace request and tape unit request has been determined. 2BT is called from the CIO monitor routine.

METHOD:

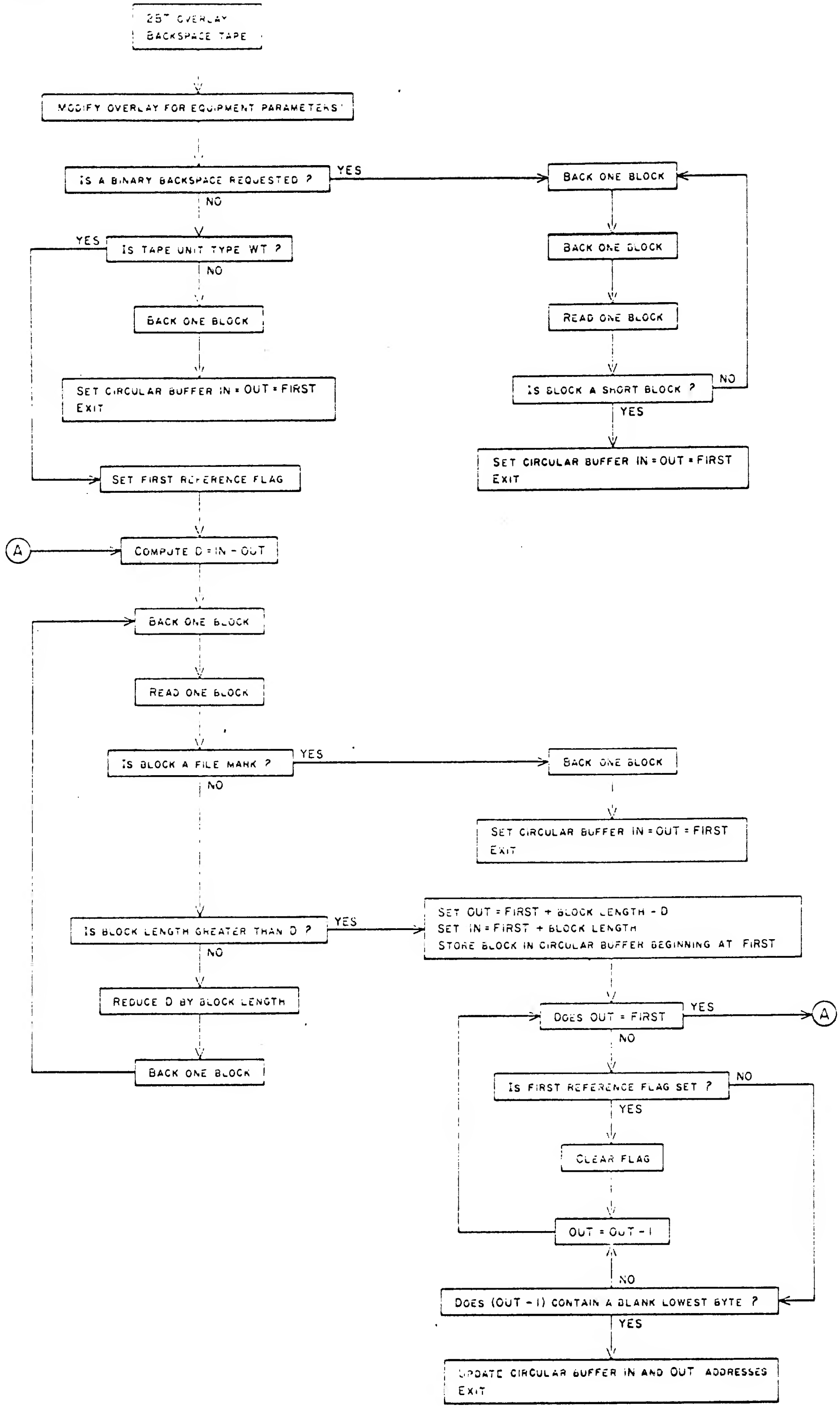
A. BINARY

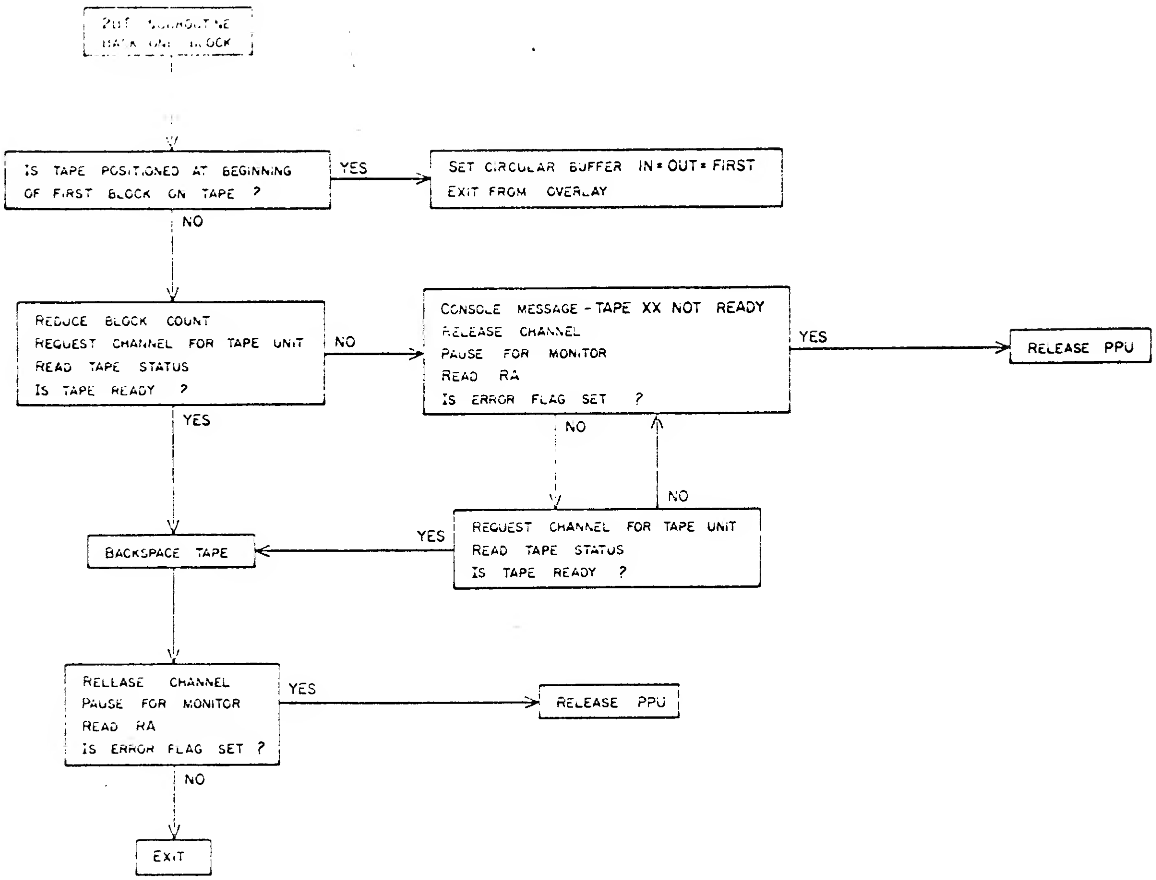
1. If a binary backspace is requested, two blocks are back-spaced and then the last one backspaced is read.
2. This block is checked for a short block; if it is short, IN and OUT are set equal to FIRST.
3. If it is not short, the backing of two blocks and reading of one is continued until a short block is found.

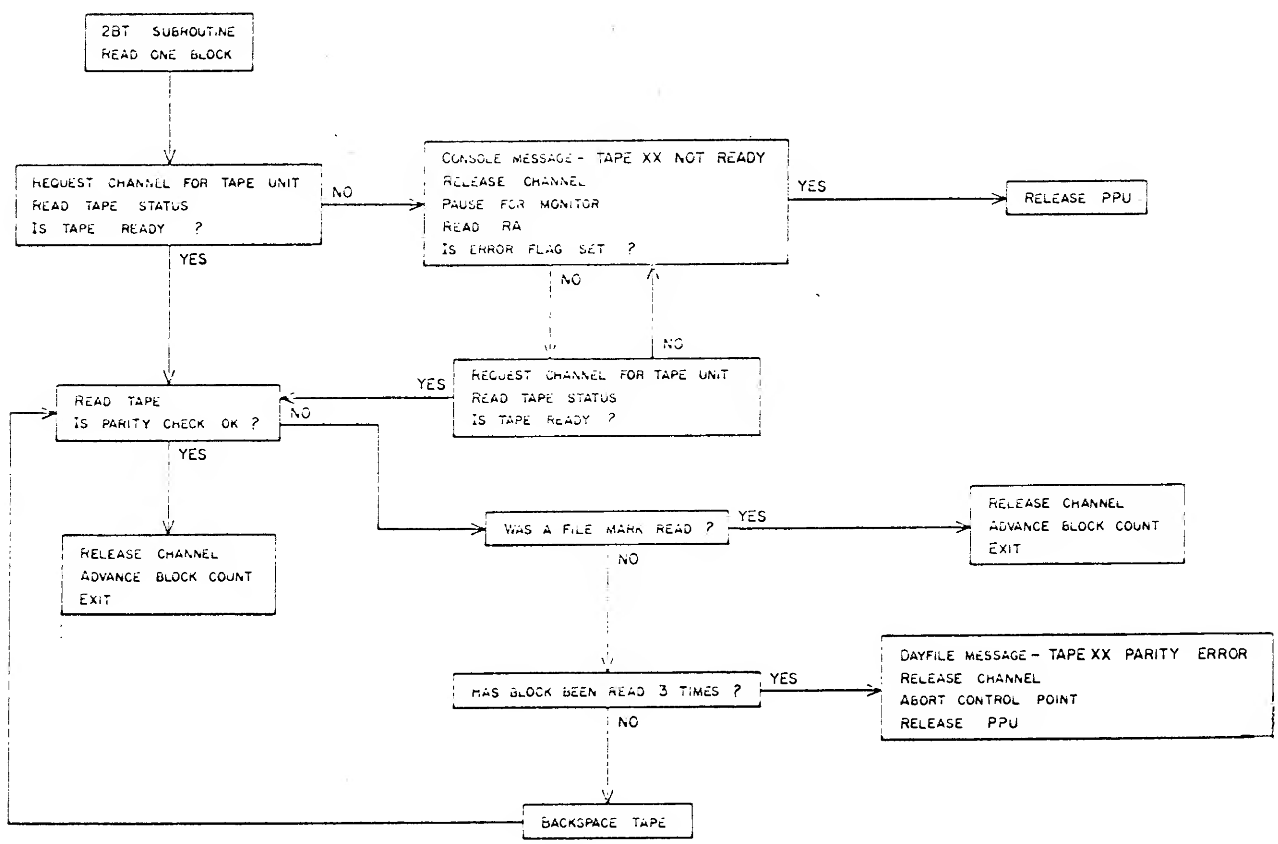
B. BCD

1. If a binary backspace is not requested, a check is made to see if tape type is WT.
2. If it is not, i.e., tape type is MT, one block is backed and IN and OUT set equeal to FIRST and then exit. This will backspace the one BCD block.
3. If the tape type is WT, a transfer is made to location 2401 (BACKSPACE CODED). A first reference flag is set and a value called D = IN-OUT is calculated.
4. One block is backed and then read into a PP buffer. If the block is a file mark, backspace back over it, set IN=OUT= FIRST and EXIT.
5. Otherwise, the block length is compared to D. If less than D, reduce D by the block length, back over the block just read, and continue the back and a read a block loop with the new value of D.
6. If the block length is greater than D, OUT is set equal to FIRST+BLOCK LENGTH - D and IN set equal to FIRST+BLOCK LENGTH, the last block read is then stored into the buffer beginn-ing at FIRST. The action just taken means the first block backed over which is small enough to fit in the buffer is stored into the buffer.
7. A check is then made to see if OUT=FIRST. If yes, the loop of calculating D and backspacing is done again. This hap-pens when the first D calculated was zero, i.e., IN=OUT, and the record backspaced and read was a four byte record.

8. If OUT does not equal FIRST, the first reference flag is checked. If it is set, it is cleared set OUT=OUT-1 and OUT is checked against FIRST. If OUT=FIRST, a one word block was read and the compute D loop is done again.
9. Once the first reference flag is cleared, a check is made to see if the contents of (OUT-1) contains a blank lowest byte. If not, set OUT=OUT-1 and recheck if OUT=FIRST.
10. If (OUT-1) has a blank lowest byte, IN and OUT are set, and EXIT taken.
11. All of this action causes OUT to be backspaced down the buffer one coded card image.
12. It should be understood that the tape will be backspaced but a read is not needed to get the backspaced record into the buffer.


257 OVERLAY
BACKSPACE TAPE
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
IS A BINARY BACKSPACE REQUESTED ?
YES
NO
BACK ONE BLOCK
BACK ONE BLOCK
READ ONE BLOCK
IS BLOCK A SHORT BLOCK ?
NO
YES
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT
IS TAPE UNIT TYPE WT ?
YES
NO
BACK ONE BLOCK
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT
SET FIRST REFERENCE FLAG
A
COMPUTE D = IN - OUT
BACK ONE BLOCK
READ ONE BLOCK
IS BLOCK A FILE MARK ?
YES
NO
BACK ONE BLOCK
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT
IS BLOCK LENGTH GREATER THAN D ?
YES
NO
SET OUT = FIRST + BLOCK LENGTH - D
SET IN = FIRST + BLOCK LENGTH
STORE BLOCK IN CIRCULAR BUFFER BEGINNING AT FIRST
REDUCE D BY BLOCK LENGTH
BACK ONE BLOCK
DOES OUT = FIRST
YES
A
NO
IS FIRST REFERENCE FLAG SET ?
NO
YES
CLEAR FLAG
OUT = OUT - 1
NO
DOES (OUT - 1) CONTAIN A BLANK LOWEST BYTE ?
YES
UPDATE CIRCULAR BUFFER IN AND OUT ADDRESSES
EXIT


IS TAPE POSITIONED AT BEGINNING OF FIRST BLOCK ON TAPE ?
YES
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT FROM OVERLAY
NO
REDUCE BLOCK COUNT
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
NO
CONSOLE MESSAGE - TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
YES
YES
BACKSPACE TAPE
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
EXIT


2BT SUBROUTINE
READ ONE BLOCK
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
NO
CONSOLE MESSAGE - TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
YES
YES
READ TAPE
IS PARITY CHECK OK ?
NO
YES
RELEASE CHANNEL
ADVANCE BLOCK COUNT
EXIT
WAS A FILE MARK READ ?
YES
RELEASE CHANNEL
ADVANCE BLOCK COUNT
EXIT
NO
HAS BLOCK BEEN READ 3 TIMES ?
YES
DAYFILE MESSAGE - TAPE XX PARITY ERROR
RELEASE CHANNEL
ABORT CONTROL POINT
RELEASE PPU
NO
BACKSPACE TAPE

ROUTINE: 2EF -- Process Error Flag

PURPOSE: To determine type of error and set up to execute the group of control cards after the EXIT. statement if one exits.

GENERAL: 2EF is called by the Advance Job routine (1AJ) when the error flag is sensed set (non-zero).

METHOD:

1. Read the control point status word from CP(20). Clears the error flag and stores status back to CP(20).

2. Uses the error flag to pick up address of error message. (Error Flag 1- Time Limit, 2- Arithmetic Error, 5- PP Call Error, 6- Operator Drop, 7- Track Limit)

3. Dayfile message routine is called to enter error message if the error condition was one of the above.

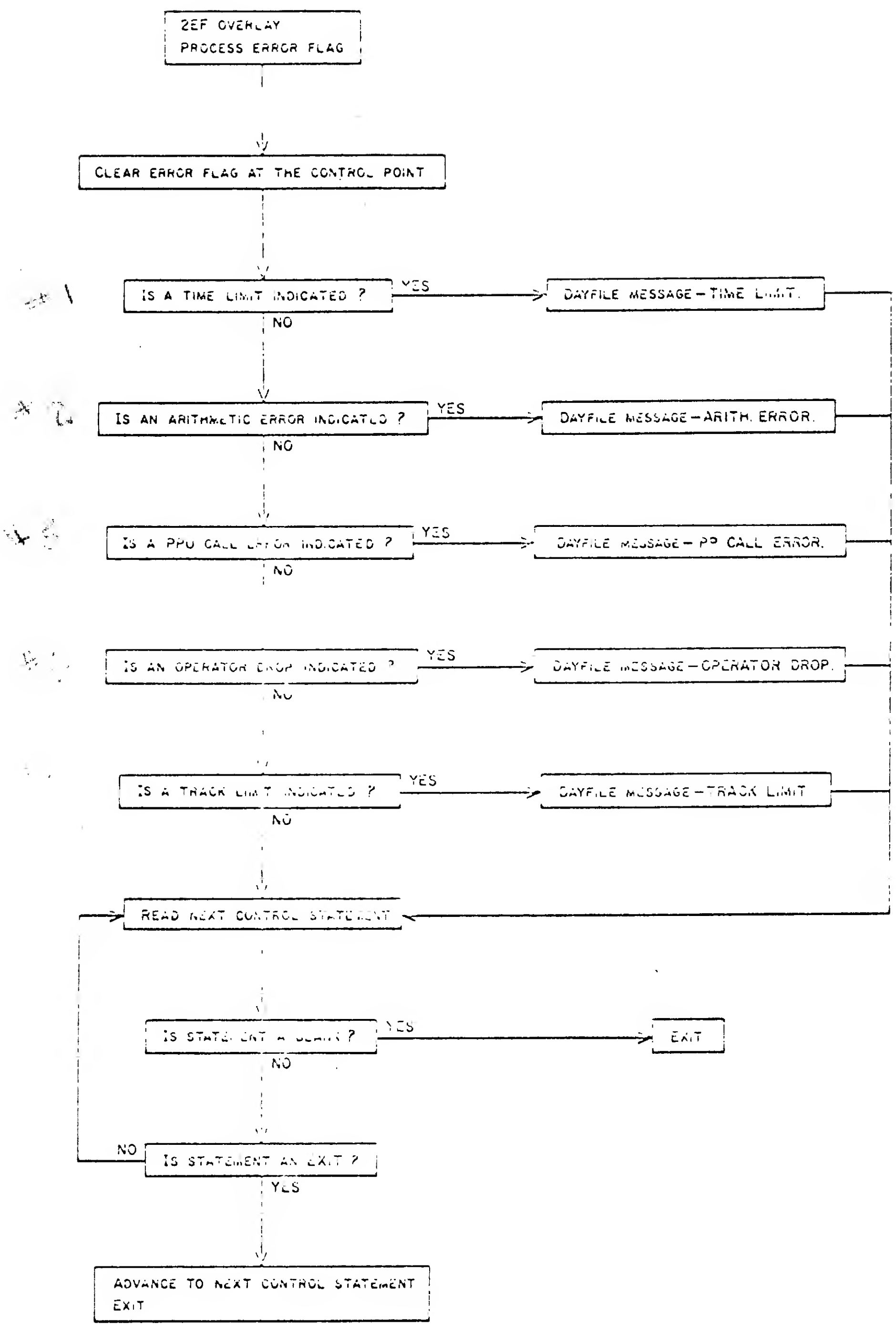
4. Control statements are searched untilthe last one is read or an EXIT. statement is encountered. The statement address at CP(21) is set to point to either the end of the statement list or the statement after the EXIT. card.

## 2EF Routines

| | | |
|---|---|---|
| 2000 | Main Program | 2030, 2100, 531 |
| 2030 | Error Table | |
| 2100 | Search for Exit | |

## Direct Core Cells

| | | |
|---|---|---|
| 2000 | P74 | CP address |
| | P10/14 | CP status |
| | P01 | Error Flag |
| 2100 | P74 | CP address |
| | P20/24 | Next statement address |
| | P10/14 | CP status |


2EF OVERLAY
PROCESS ERROR FLAG
CLEAR ERROR FLAG AT THE CONTROL POINT
IS A TIME LIMIT INDICATED ?
YES
DAYFILE MESSAGE – TIME LIMIT.
NO
IS AN ARITHMETIC ERROR INDICATED ?
YES
DAYFILE MESSAGE – ARITH. ERROR.
NO
IS A PPU CALL ERROR INDICATED ?
YES
DAYFILE MESSAGE – PP CALL ERROR.
NO
IS AN OPERATOR DROP INDICATED ?
YES
DAYFILE MESSAGE – OPERATOR DROP.
NO
IS A TRACK LIMIT INDICATED ?
YES
DAYFILE MESSAGE – TRACK LIMIT
NO
READ NEXT CONTROL STATEMENT
IS STATEMENT A BLANK ?
YES
EXIT
NO
NO
IS STATEMENT AN EXIT ?
YES
ADVANCE TO NEXT CONTROL STATEMENT
EXIT

ROUTINE: 2LP -- PRINT

PURPOSE: To transfer data from the circular buffer to the line printer.

GENERAL: 2LP is called by the CIO Write Function routine once the file type has been determined to be a line printer.

METHOD:

1. A check is made for data left in the buffer. If there is none and the end-of-record was requested, IN and OUT are set equal to first, and EXIT is taken.

2. If there is data, a word is read up and copied into the print line buffer. If the lowest byte of the word is not zero and 120 characters have not been assembled, another word is fetched.

3. If either a zero byte is found or 120 characters have been assembled, a transfer is made to location 2150 (PRINT LINE). This subroutine finds an available printer and prints the line.

4. Three characters are checked in the first character position for carriage control:

   (0) - advance paper one extra line after printing.

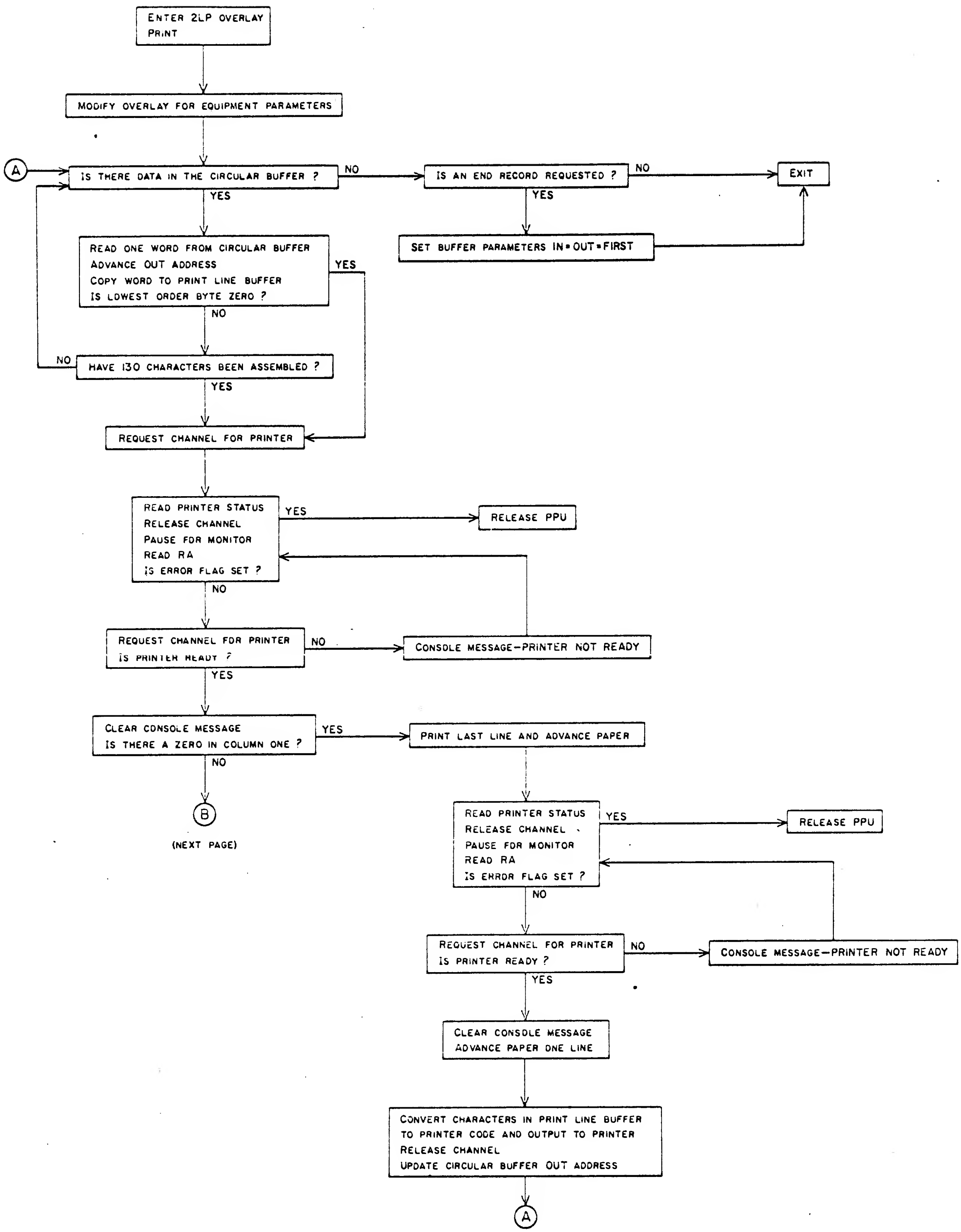
   (1) - advance to top of form after printing line.

   (+) - print the last line but do not advance the paper.

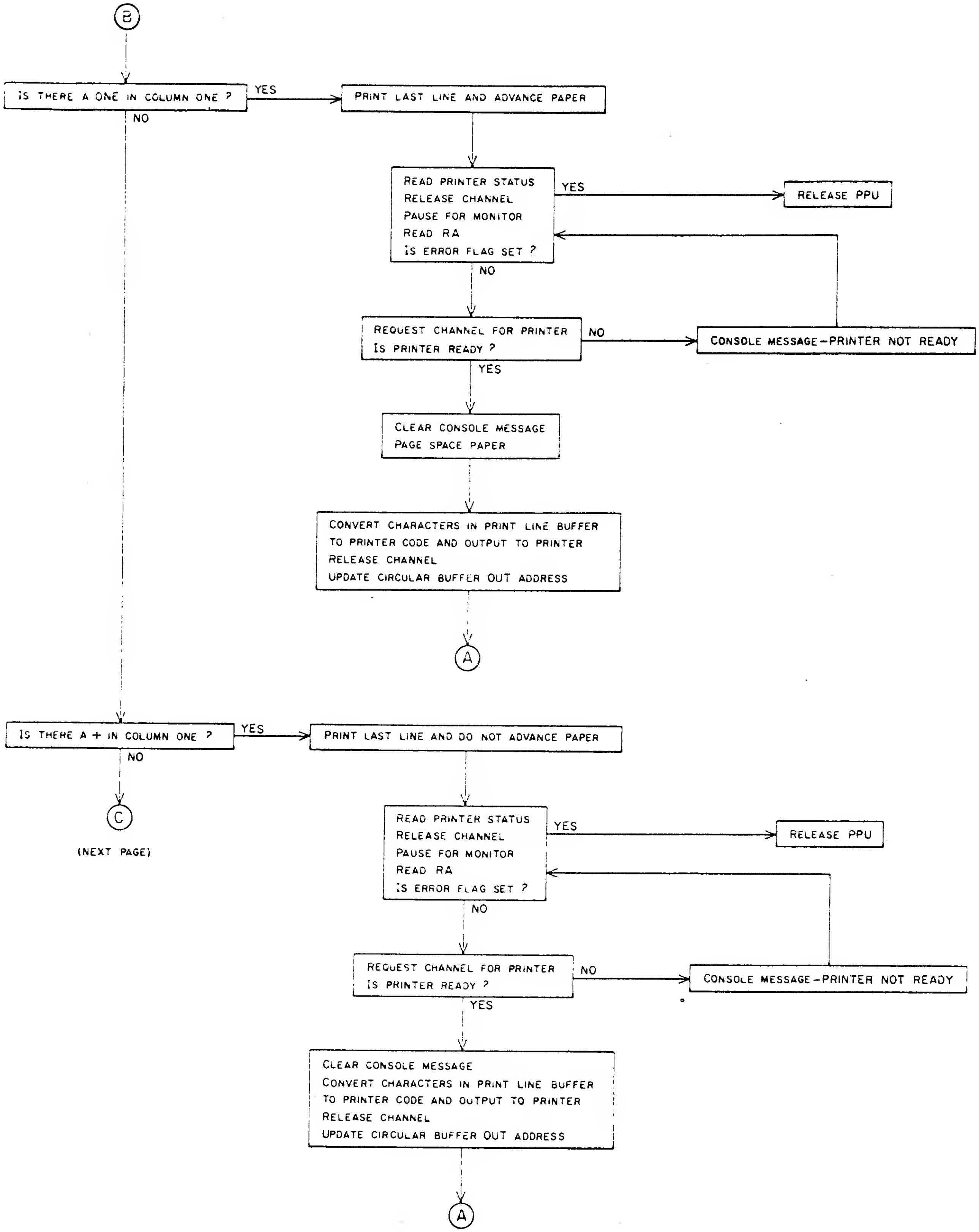
5. If there is a 7X code in column one, the last line is printed, and then printer carriage control X is selected.

6. If none of the above mentioned codes are in column one, the line is printed and the paper advanced.

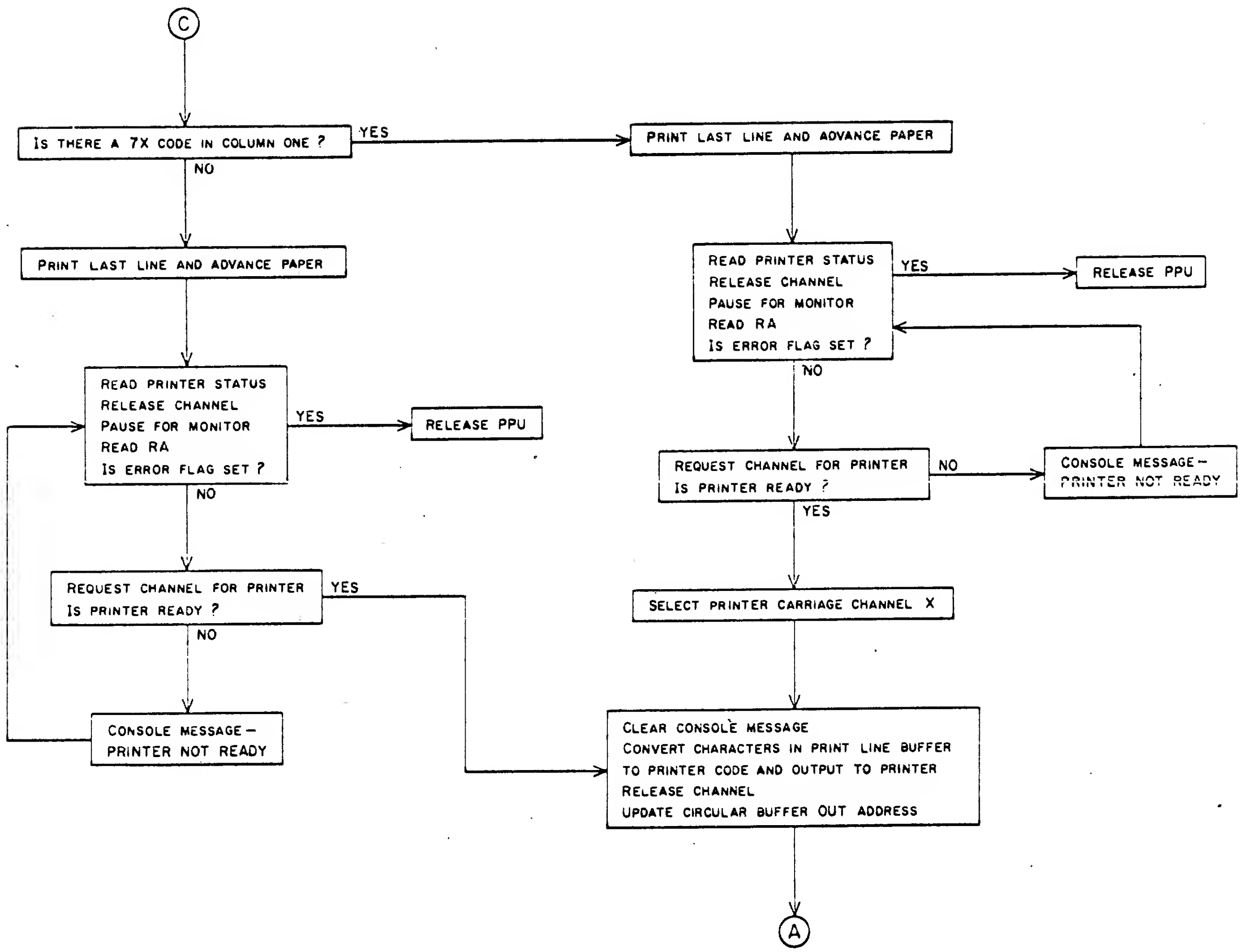
7. 2LP then returns to its beginning routine to check if any data is left in the buffer.


ENTER 2LP OVERLAY PRINT
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
A
IS THERE DATA IN THE CIRCULAR BUFFER ?
NO
IS AN END RECORD REQUESTED ?
NO
EXIT
YES
SET BUFFER PARAMETERS IN=OUT=FIRST
YES
READ ONE WORD FROM CIRCULAR BUFFER
ADVANCE OUT ADDRESS
COPY WORD TO PRINT LINE BUFFER
IS LOWEST ORDER BYTE ZERO ?
YES
NO
HAVE 130 CHARACTERS BEEN ASSEMBLED ?
NO
YES
REQUEST CHANNEL FOR PRINTER
READ PRINTER STATUS
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
REQUEST CHANNEL FOR PRINTER
IS PRINTER READY ?
NO
CONSOLE MESSAGE—PRINTER NOT READY
YES
CLEAR CONSOLE MESSAGE
IS THERE A ZERO IN COLUMN ONE ?
YES
PRINT LAST LINE AND ADVANCE PAPER
NO
B
(NEXT PAGE)
READ PRINTER STATUS
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
REQUEST CHANNEL FOR PRINTER
IS PRINTER READY ?
NO
CONSOLE MESSAGE—PRINTER NOT READY
YES
CLEAR CONSOLE MESSAGE
ADVANCE PAPER ONE LINE
CONVERT CHARACTERS IN PRINT LINE BUFFER
TO PRINTER CODE AND OUTPUT TO PRINTER
RELEASE CHANNEL
UPDATE CIRCULAR BUFFER OUT ADDRESS
A

(2LP CONTINUED)



(2LP CONTINUED)



ROUTINE: 2PC -- Punch Cards

PURPOSE: To punch either binary or Hollerith cards.

GENERAL: 2PC is called by the CIO Write Function routine once the file type has been determined to be a card punch.

METHOD:

1. A check is made for a request to punch Hollerith. A return jump is made to either PUNCH BINARY or PUNCH BCD.

   A. PUNCH BINARY

      1. If there is enough data for a full card, the punch buffer is cleared for 15 words.

         a) The data is transferred to the punch buffer.

         b) The card length is set in column one.

         c) The checksum set in column two.

         d) The card count is advanced.

         e) The card count is entered in column 80.

      2. A channel is then requested, with a "PUNCH NOT READY" message displayed if needed. If the punch is ready, a card is punched, the channel released, OUT is updated, and a check for the error flag in RA is made.

      3. If an error exists, the PPU is released.

      4. If there is no error, a check is again made to see if there is enough data for a full card. If there is not enough data for a full card and an end-of-record is selected, the partial card will be punched.

      5. If there is no data, a 7-8-9 card is punched, and IN and OUT are set equal to FIRST.

   B. PUNCH ~~BINARY~~ BCD

      1. The punch buffer is cleared for 80 characters.

      2. If there is at least one word left in the buffer, the word is converted into 10 Hollerith characters.

      3. A check is always made to see if 80 characters have been assembled. If not, a check for a lowest order byte of zero is made. If it is not present, another word is assembled.

4. If either 80 characters have been assembled or a zero byte is found, the card is punched.

5. The error flag is then checked in RA and the PP is released if an error exists. If there is no error, OUT is updated, the card count advanced, and a return is made to convert another 80 characters.

6. If there is not another full word in the buffer and a file mark is requested:

    a) The card count is cleared.

    b) A 6-7-8-9 card is punched.

    c) IN and OUT set equal to FIRST, and

    d) An EXIT taken.

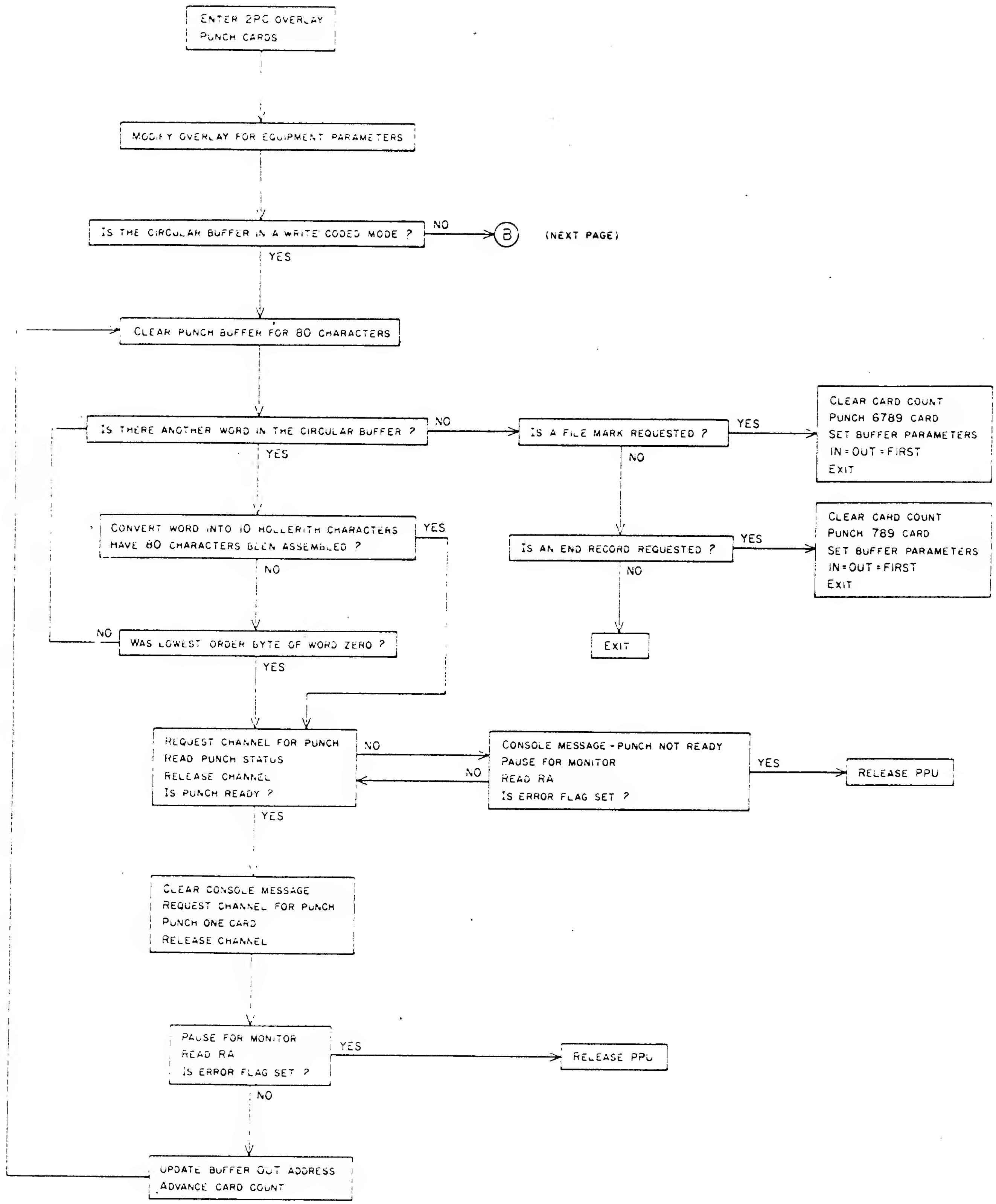
7. If an end-of-record is requested,

    a) A 7-8-9 card is punched.

    b) The card count is cleared.

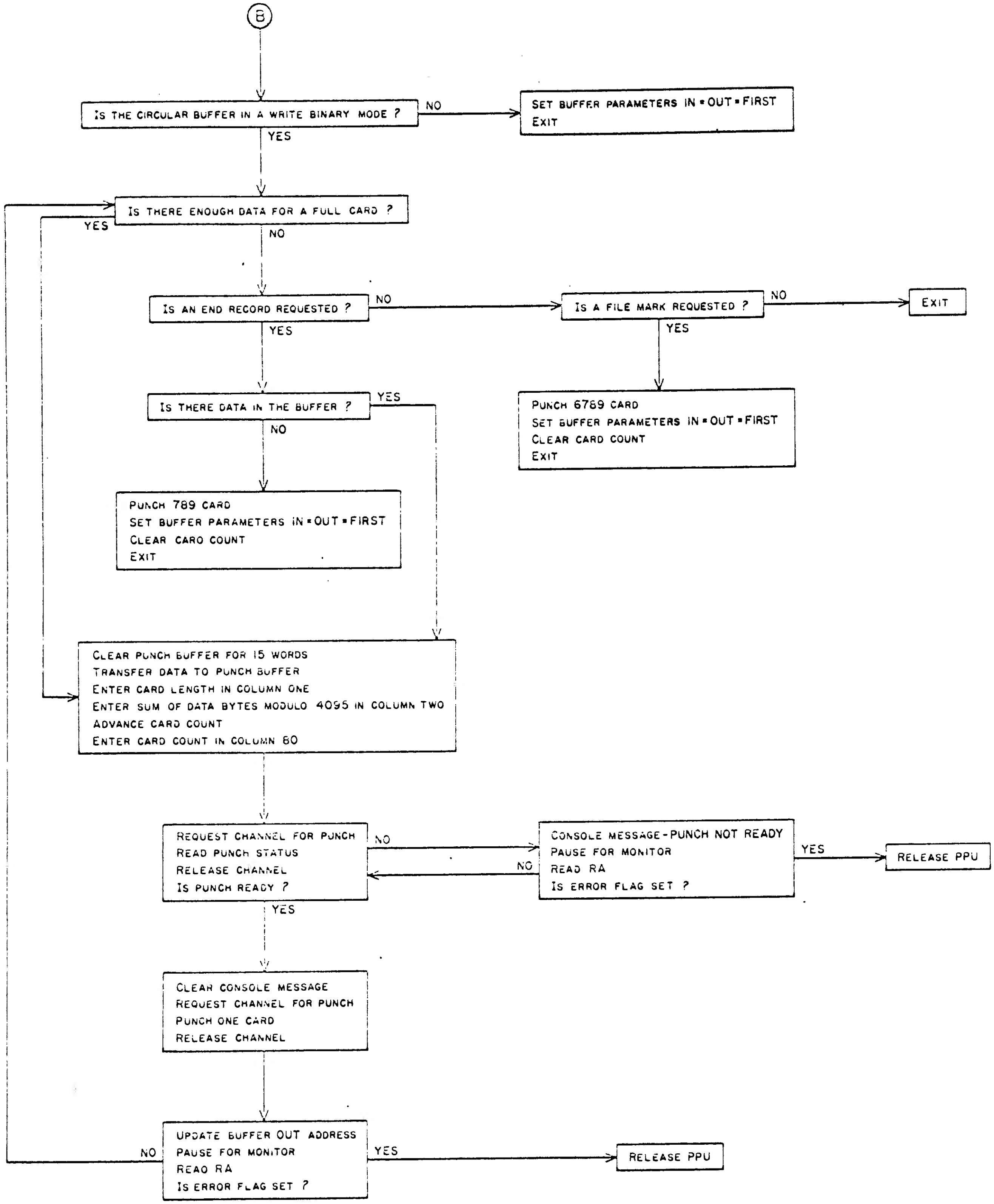
    c) IN and OUT set equal to FIRST, and

    d) EXIT taken.



ENTER 2PC OVERLAY
PUNCH CARDS
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
IS THE CIRCULAR BUFFER IN A WRITE CODED MODE ?
NO
B
(NEXT PAGE)
YES
CLEAR PUNCH BUFFER FOR 80 CHARACTERS
IS THERE ANOTHER WORD IN THE CIRCULAR BUFFER ?
NO
IS A FILE MARK REQUESTED ?
YES
CLEAR CARD COUNT
PUNCH 6789 CARD
SET BUFFER PARAMETERS
IN = OUT = FIRST
EXIT
NO
YES
CONVERT WORD INTO 10 HOLLERITH CHARACTERS
HAVE 80 CHARACTERS BEEN ASSEMBLED ?
YES
IS AN END RECORD REQUESTED ?
YES
CLEAR CARD COUNT
PUNCH 789 CARD
SET BUFFER PARAMETERS
IN = OUT = FIRST
EXIT
NO
NO
EXIT
NO
WAS LOWEST ORDER BYTE OF WORD ZERO ?
YES
REQUEST CHANNEL FOR PUNCH
READ PUNCH STATUS
RELEASE CHANNEL
IS PUNCH READY ?
NO
CONSOLE MESSAGE - PUNCH NOT READY
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
YES
RELEASE PPU
YES
CLEAR CONSOLE MESSAGE
REQUEST CHANNEL FOR PUNCH
PUNCH ONE CARD
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
UPDATE BUFFER OUT ADDRESS
ADVANCE CARD COUNT

(2PC OVERLAY CONTINUED)



ROUTINE: 2RC - Read Cards

PURPOSE: To read cards from the card reader and process them either as binary or BCD cards.

GENERAL: 2RC is called by the CIO Read Function routine once the file type has been determined to be a card reader.

METHOD:

1. If the End-of-Job flag is set, 2RC clears the flag, sets the file mark and exits.

2. A check is made to see if the buffer has room for 15 words of input. If not, an EXIT is taken and no read is performed.

3. A return jump is taken to READ NEXT CARD which requests the correct channel, makes sure the reader is ready, and reads the next card.

4. Once a card is read, the card count is advanced in the FST entry and a check is made for 7-8-9 punches in column one. If there are only 7-9 punches, a transfer is made to PROCESS BINARY CARD. If neither condition exists, PROCESS HOLLERITH CARD is given control.

5. After the card is processed, the IN address of the central memory buffer is incremented by the number of words read.

6. Another card is then read if the buffer length allows it and there are no errors.

7. If a 7-8-9 card was found, an end-of-record indicator is set and the card count is cleared. An EXIT is then made.

8. If a 6-7-8-9 card was found:
   a) and the last record was not complete, the End-of-Job flag is set along with End-of-record. The next time through 2RC, the EOJ flag will be cleared and a file mark will be written.

   b) and the last record was complete, the file mark indicator is set if the buffer is empty.

   c) and the last record was complete, the EOJ flag and End-of-record indicator are set if the buffer is not empty.

A. PROCESS BINARY CARD

1. The number of significant columns is determined from the word count in column one.

2. If there is a correction punch in column one, the significant words are copied into the circular buffer, IN is advanced and an EXIT taken.

3. Otherwise, the checksum is cleared and the column index is set to 2. Each significant column is then added to the checksum module 4095. If the checksum is zero, the significant words are copied into the buffer and IN is advanced.

4. If the checksum is not zero, a binary card error is displayed. After a 4 second delay, a check is made to see if the card reader is ready. If it is not, then the operator is given a chance to reread the card.

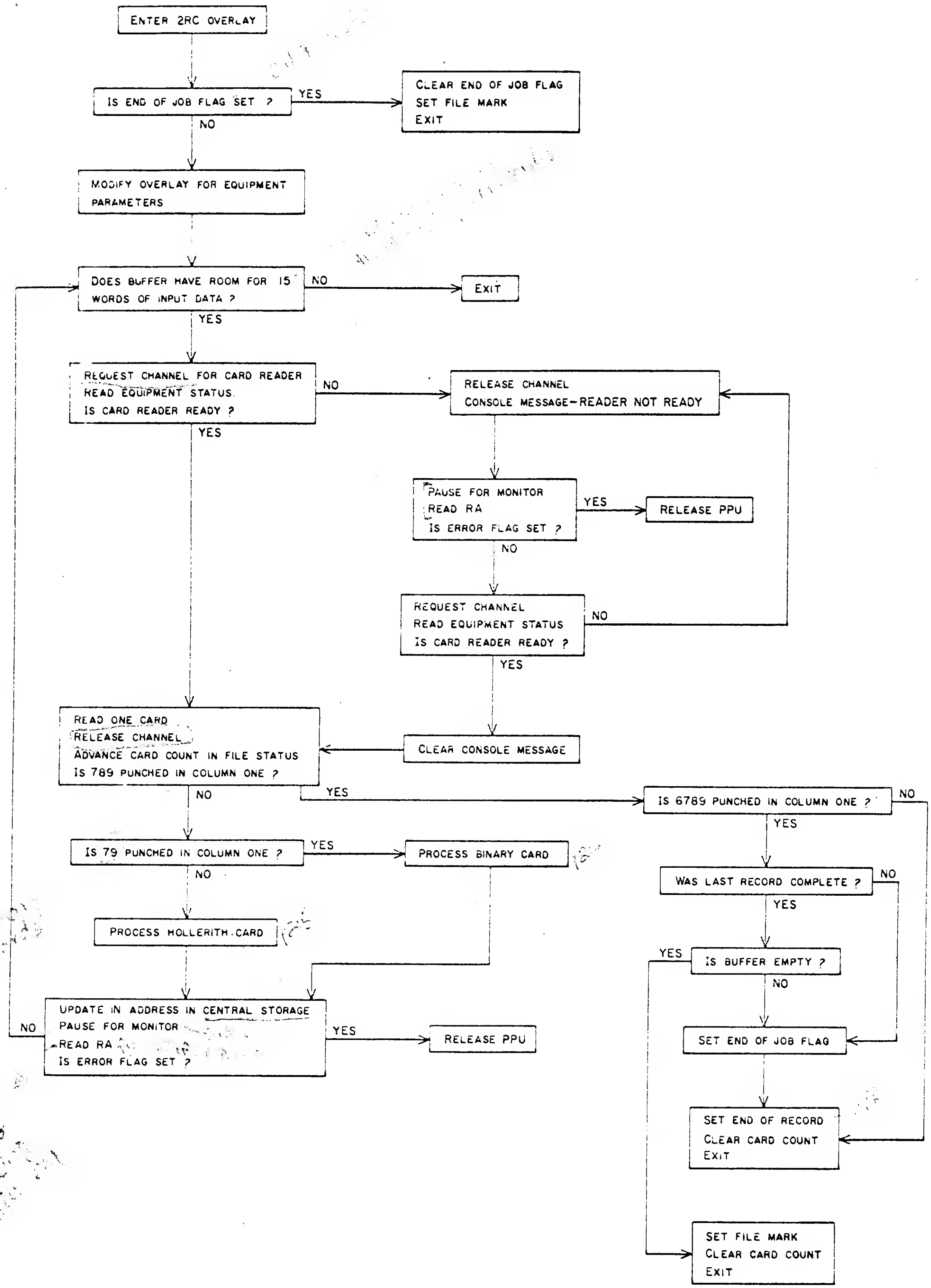
5. If the reader is ready, a check of the error flag in RA is made. If the error flag is not set, then the binary card error is displayed again.

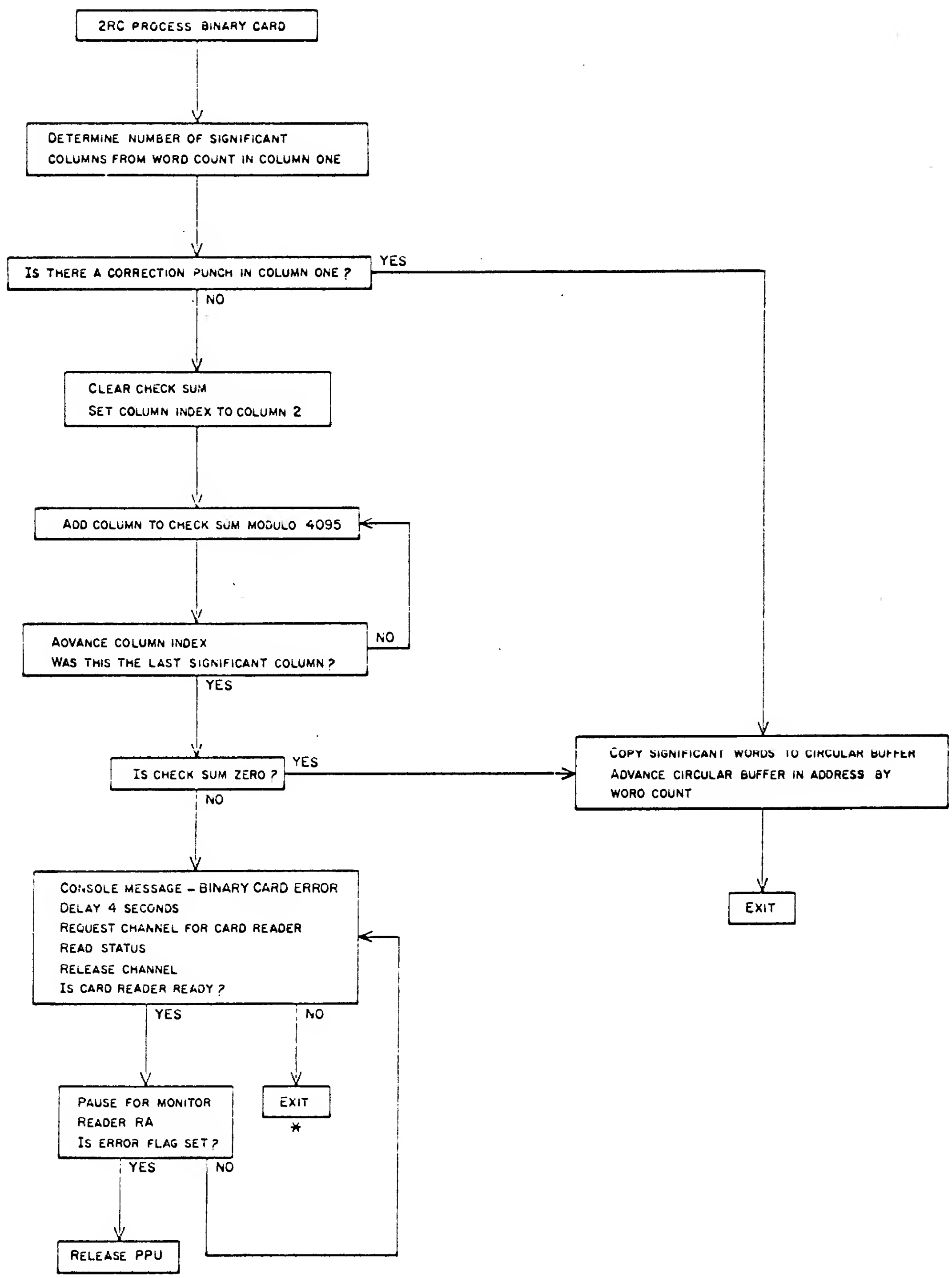
B. PROCESS HOLLERITH CARD

1. The last significant column is determined.

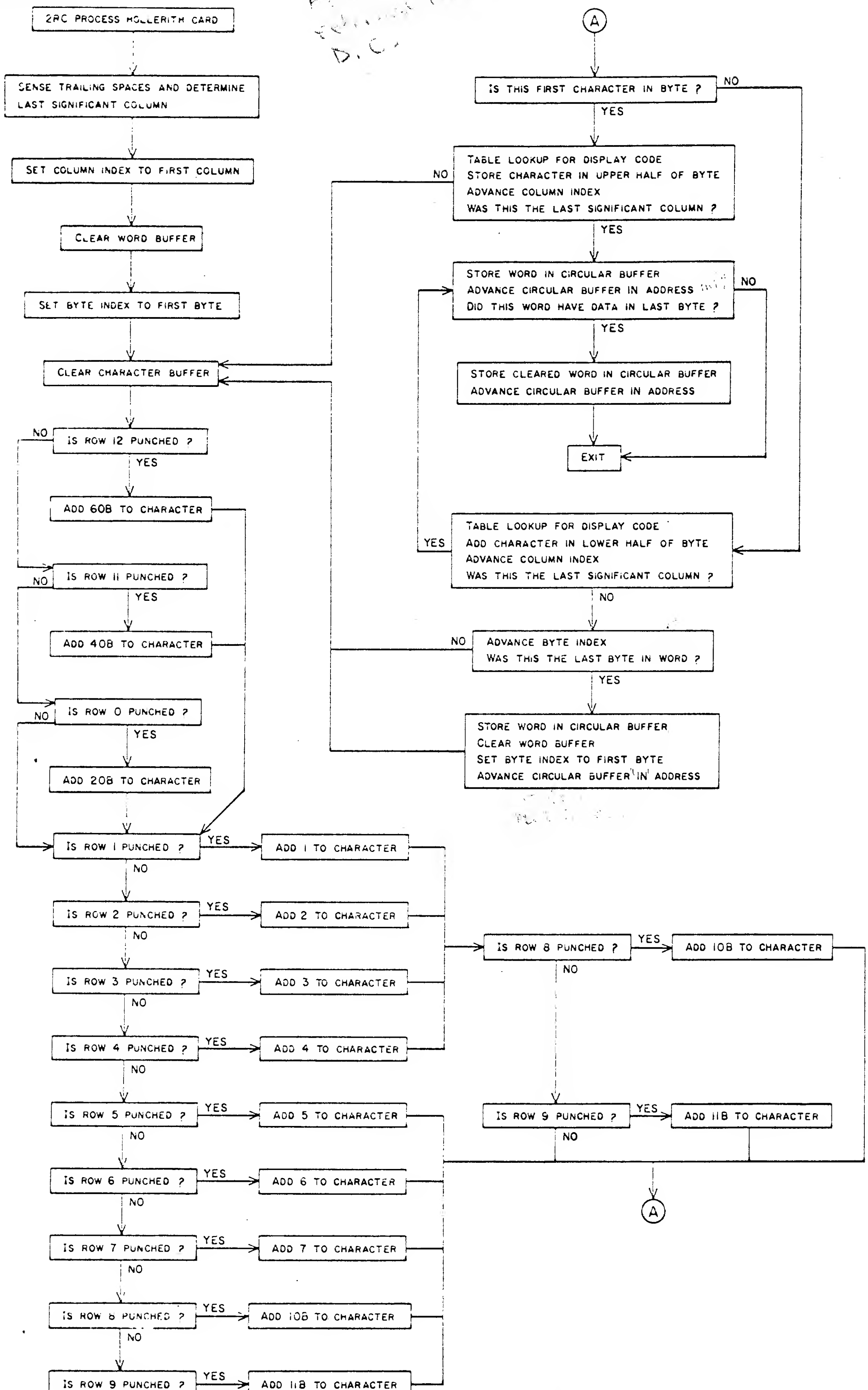
2. A table look-up is then done on each character to change the Hollerith character into display code. The significant characters are stored in the buffer by advancing IN.

3. If the last word's last byte has significant data, a cleared word is stored after it. If not, the last byte will be cleared.


ENTER 2RC OVERLAY
IS END OF JOB FLAG SET ?
YES
CLEAR END OF JOB FLAG
SET FILE MARK
EXIT
NO
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
DOES BUFFER HAVE ROOM FOR 15 WORDS OF INPUT DATA ?
NO
EXIT
YES
REQUEST CHANNEL FOR CARD READER
READ EQUIPMENT STATUS.
IS CARD READER READY ?
NO
RELEASE CHANNEL
CONSOLE MESSAGE–READER NOT READY
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
REQUEST CHANNEL
READ EQUIPMENT STATUS
IS CARD READER READY ?
NO
YES
CLEAR CONSOLE MESSAGE
YES
READ ONE CARD
RELEASE CHANNEL
ADVANCE CARD COUNT IN FILE STATUS
IS 789 PUNCHED IN COLUMN ONE ?
YES
IS 6789 PUNCHED IN COLUMN ONE ?
NO
YES
WAS LAST RECORD COMPLETE ?
NO
YES
IS BUFFER EMPTY ?
YES
NO
SET END OF JOB FLAG
SET END OF RECORD
CLEAR CARD COUNT
EXIT
SET FILE MARK
CLEAR CARD COUNT
EXIT
NO
IS 79 PUNCHED IN COLUMN ONE ?
YES
PROCESS BINARY CARD
NO
PROCESS HOLLERITH CARD
UPDATE IN ADDRESS IN CENTRAL STORAGE
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
YES
RELEASE PPU




* THIS PATH PROVIDES AN OPPORTUNITY FOR THE OPERATOR TO REREAD THE FAULTY CARD


2RC PROCESS HOLLERITH CARD
SENSE TRAILING SPACES AND DETERMINE LAST SIGNIFICANT COLUMN
SET COLUMN INDEX TO FIRST COLUMN
CLEAR WORD BUFFER
SET BYTE INDEX TO FIRST BYTE
CLEAR CHARACTER BUFFER
IS ROW 12 PUNCHED ?
ADD 60B TO CHARACTER
IS ROW 11 PUNCHED ?
ADD 40B TO CHARACTER
IS ROW 0 PUNCHED ?
ADD 20B TO CHARACTER
IS ROW 1 PUNCHED ?
ADD 1 TO CHARACTER
IS ROW 2 PUNCHED ?
ADD 2 TO CHARACTER
IS ROW 3 PUNCHED ?
ADD 3 TO CHARACTER
IS ROW 4 PUNCHED ?
ADD 4 TO CHARACTER
IS ROW 5 PUNCHED ?
ADD 5 TO CHARACTER
IS ROW 6 PUNCHED ?
ADD 6 TO CHARACTER
IS ROW 7 PUNCHED ?
ADD 7 TO CHARACTER
IS ROW 8 PUNCHED ?
ADD 10B TO CHARACTER
IS ROW 9 PUNCHED ?
ADD 11B TO CHARACTER
IS ROW 8 PUNCHED ?
ADD 10B TO CHARACTER
IS ROW 9 PUNCHED ?
ADD 11B TO CHARACTER
A
IS THIS FIRST CHARACTER IN BYTE ?
TABLE LOOKUP FOR DISPLAY CODE
STORE CHARACTER IN UPPER HALF OF BYTE
ADVANCE COLUMN INDEX
WAS THIS THE LAST SIGNIFICANT COLUMN ?
STORE WORD IN CIRCULAR BUFFER
ADVANCE CIRCULAR BUFFER IN ADDRESS
DID THIS WORD HAVE DATA IN LAST BYTE ?
STORE CLEARED WORD IN CIRCULAR BUFFER
ADVANCE CIRCULAR BUFFER IN ADDRESS
EXIT
TABLE LOOKUP FOR DISPLAY CODE
ADD CHARACTER IN LOWER HALF OF BYTE
ADVANCE COLUMN INDEX
WAS THIS THE LAST SIGNIFICANT COLUMN ?
ADVANCE BYTE INDEX
WAS THIS THE LAST BYTE IN WORD ?
STORE WORD IN CIRCULAR BUFFER
CLEAR WORD BUFFER
SET BYTE INDEX TO FIRST BYTE
ADVANCE CIRCULAR BUFFER IN ADDRESS
YES
NO

ROUTINE: 2RT -- Read Tape

PURPOSE: To read binary and BCD data from magnetic tape or rewind the tape.

GENERAL: This package is called by the CIO Read Function when a magnetic tape is to be read. Control is transferred by CIO to one of three locations within 2RT:

a) READ BINARY TAPE

b) READ BCD TAPE

c) REWIND

METHOD: A. READ BINARY TAPE

1. There must be room in the buffer for a full block ($1000_8$ words) of data or no reading is done.

2. The requested tape unit status is checked. If it is not ready, the message "TAPE XX NOT READY" is sent to the control point display and no further processing is done until the tape is ready or an error flag is set.

3. One block of data is read in odd parity. If the length is less than 4 bytes (signifying noise) it is ignored and another record is read.

4. If an end-of-file was encountered, the buffer status is changed to reflect it and an EXIT is made.

5. When a parity error is encountered, the tape is backspaced one block and reread. The message "TAPE XX PARITY ERROR" is sent if the parity error still exists after 3 attempts. A pause bit is set in RA and is cleared only after "X.GO." is typed in answer to the display message.

6. When the pause bit is cleared, the bad data is stored in the buffer and a new block is read.

7. The data is read until an end-of-record or end-of-file is sensed.

B. READ BCD TAPE

1. The requested tape unit status is checked. If it is not ready, the message "TAPE XX NOT READY" is sent to the control point display and no further processing is done until the tape is ready or an error flag is set.

2. One block of data is read in even parity. If an end-of-file was encountered, the buffer status is changed to reflect it and an EXIT is made. If the length is less than 6 bytes (signifying noise), it is ignored and another record is read.

3. If a parity error is sensed, the tape is backspaced one block and reread. The message "TAPE XX PARITY ERROR" is sent if the parity error persists after 3 attempts. A pause bit is set in RA and is cleared only after "X.GO." is typed in answer to the display message.

4. When the pause bit is cleared, the normal processing continues.

5. The number of significant BCD characters is determined and trailing spaces are suppressed by a zero byte.

6. The BCD characters are converted to display code by a table look-up. A blank ($55_8$) is substituted for an illegal character.

7. The data is copied into the central memory circular buffer until a zero byte is found.

8. Only one record (120 characters) is read and then an EXIT is made.

C. REWIND/UNLOAD

1. The tape is checked for ready status and if an unload was requested the tape is rewound and then unloaded.

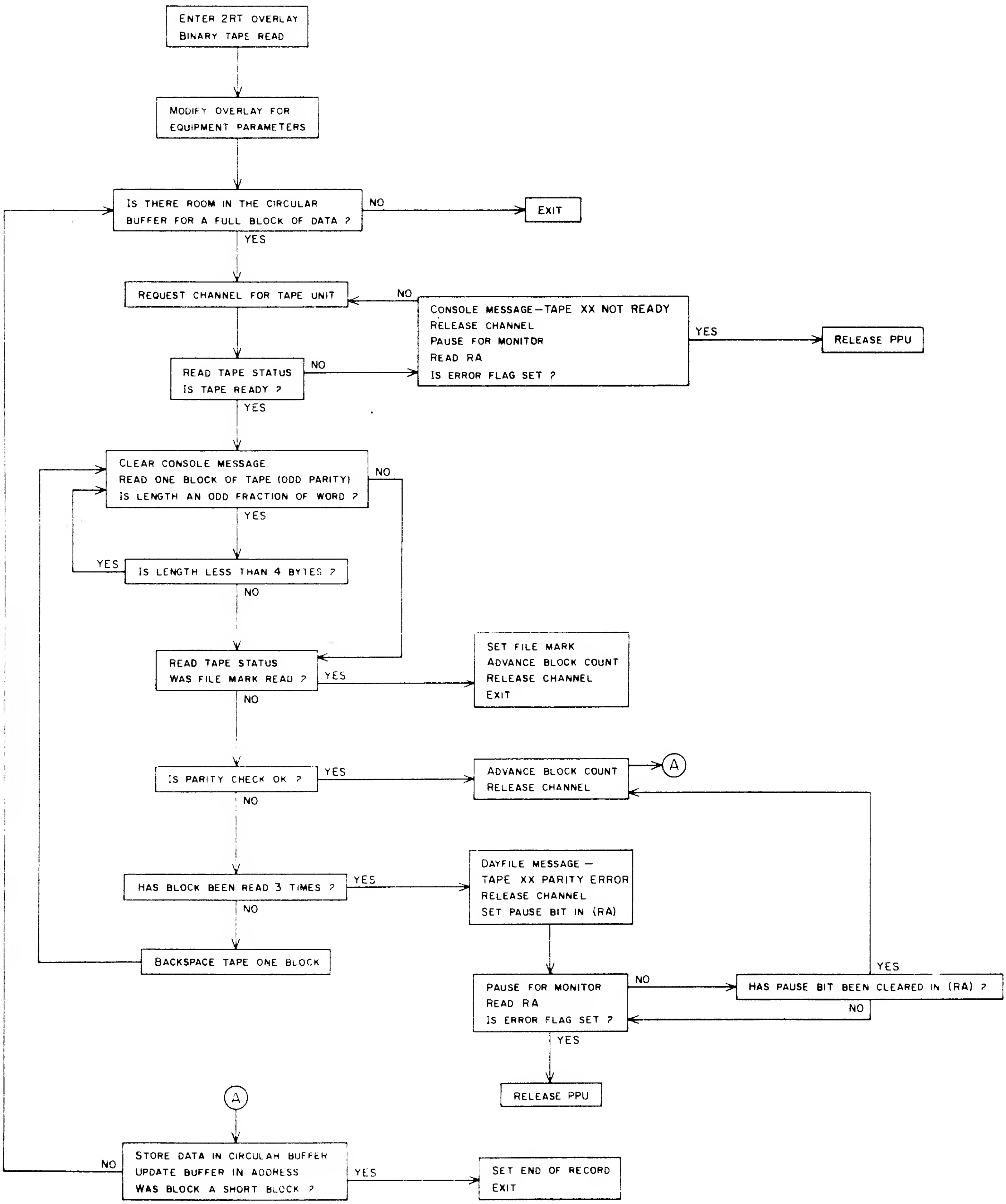
2. If only a rewind was requested, the tape is rewound.

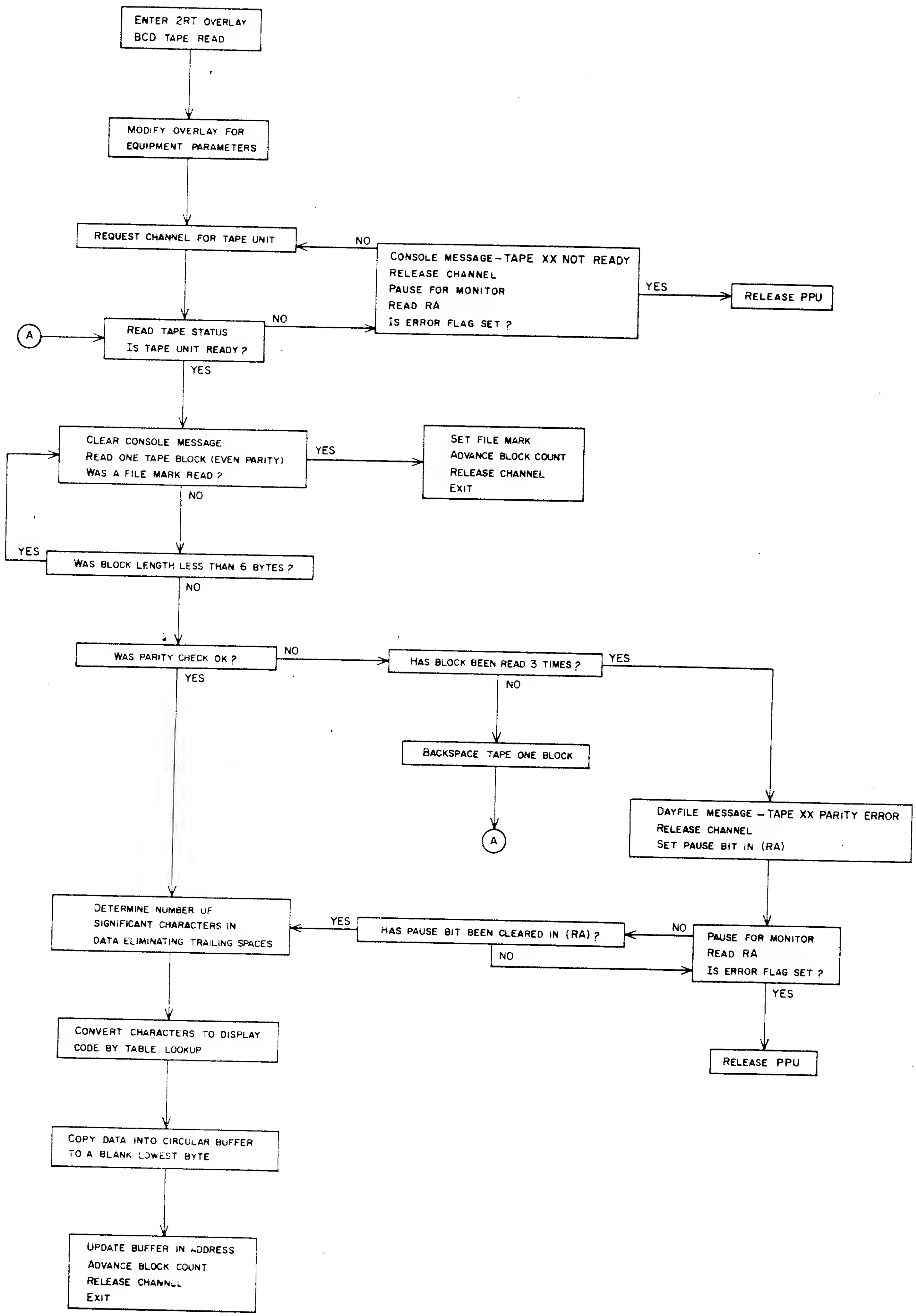
3. The block count in the FST entry is cleared and an EXIT is taken.

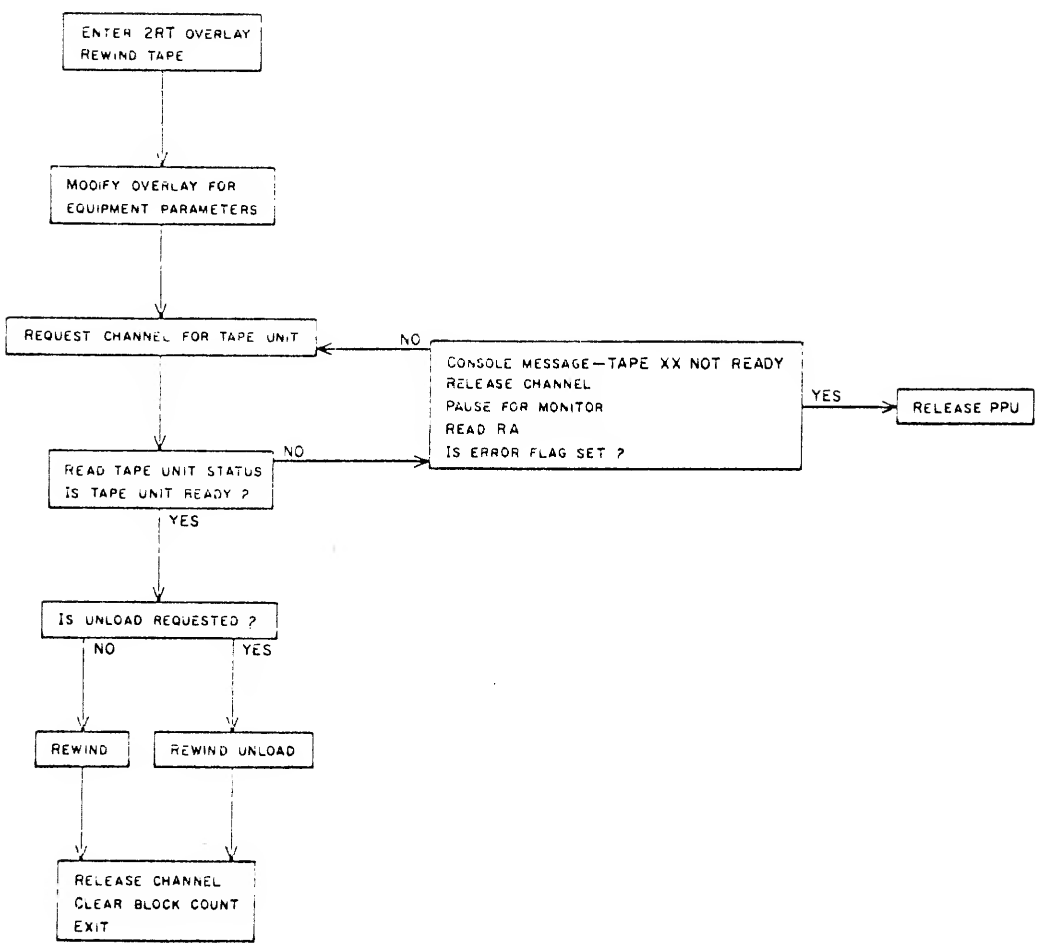
NOTES:

1. Noise records in binary is a block less than 4 bytes and in BCD less than 6 bytes.

2. BCD characters which do not have a legal display code counterpart become blanks ($55_8$).


ENTER 2RT OVERLAY
BINARY TAPE READ
MODIFY OVERLAY FOR
EQUIPMENT PARAMETERS
IS THERE ROOM IN THE CIRCULAR
BUFFER FOR A FULL BLOCK OF DATA ?
NO
EXIT
YES
REQUEST CHANNEL FOR TAPE UNIT
NO
CONSOLE MESSAGE—TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
READ TAPE STATUS
IS TAPE READY ?
NO
YES
CLEAR CONSOLE MESSAGE
READ ONE BLOCK OF TAPE (ODD PARITY)
IS LENGTH AN ODD FRACTION OF WORD ?
NO
YES
YES
IS LENGTH LESS THAN 4 BYTES ?
NO
READ TAPE STATUS
WAS FILE MARK READ ?
YES
SET FILE MARK
ADVANCE BLOCK COUNT
RELEASE CHANNEL
EXIT
NO
IS PARITY CHECK OK ?
YES
ADVANCE BLOCK COUNT
RELEASE CHANNEL
A
NO
HAS BLOCK BEEN READ 3 TIMES ?
YES
DAYFILE MESSAGE —
TAPE XX PARITY ERROR
RELEASE CHANNEL
SET PAUSE BIT IN (RA)
NO
BACKSPACE TAPE ONE BLOCK
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
HAS PAUSE BIT BEEN CLEARED IN (RA) ?
YES
NO
YES
RELEASE PPU
A
STORE DATA IN CIRCULAR BUFFER
UPDATE BUFFER IN ADDRESS
WAS BLOCK A SHORT BLOCK ?
NO
YES
SET END OF RECORD
EXIT


ENTER 2RT OVERLAY
BCD TAPE READ
MODIFY OVERLAY FOR
EQUIPMENT PARAMETERS
REQUEST CHANNEL FOR TAPE UNIT
NO
CONSOLE MESSAGE-TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
A
READ TAPE STATUS
IS TAPE UNIT READY ?
NO
YES
CLEAR CONSOLE MESSAGE
READ ONE TAPE BLOCK (EVEN PARITY)
WAS A FILE MARK READ ?
YES
SET FILE MARK
ADVANCE BLOCK COUNT
RELEASE CHANNEL
EXIT
NO
YES
WAS BLOCK LENGTH LESS THAN 6 BYTES ?
NO
WAS PARITY CHECK OK ?
NO
HAS BLOCK BEEN READ 3 TIMES ?
YES
NO
BACKSPACE TAPE ONE BLOCK
A
DAYFILE MESSAGE - TAPE XX PARITY ERROR
RELEASE CHANNEL
SET PAUSE BIT IN (RA)
YES
DETERMINE NUMBER OF
SIGNIFICANT CHARACTERS IN
DATA ELIMINATING TRAILING SPACES
YES
HAS PAUSE BIT BEEN CLEARED IN (RA) ?
NO
NO
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
CONVERT CHARACTERS TO DISPLAY
CODE BY TABLE LOOKUP
COPY DATA INTO CIRCULAR BUFFER
TO A BLANK LOWEST BYTE
UPDATE BUFFER IN ADDRESS
ADVANCE BLOCK COUNT
RELEASE CHANNEL
EXIT



ENTER 2RT OVERLAY
REWIND TAPE
MODIFY OVERLAY FOR
EQUIPMENT PARAMETERS
REQUEST CHANNEL FOR TAPE UNIT
NO
CONSOLE MESSAGE—TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
READ TAPE UNIT STATUS
IS TAPE UNIT READY ?
YES
IS UNLOAD REQUESTED ?
NO
YES
REWIND
REWIND UNLOAD
RELEASE CHANNEL
CLEAR BLOCK COUNT
EXIT

ROUTINE: 2TJ -- Translate Job Card

PURPOSE: To check the parameters on the job card for errors and assemble the values for use by other routines.

GENERAL: 2TJ is called by 1BJ, 1LJ, or 1LT. The job card is read from the control card buffer located in the control point area. Upper entry to 2TJ, the buffer parameters are passed through the PP's direct core cells. All job card parameters except the job name are converted from display code to binary.

METHOD:

1. If the circular buffer contains more than 95 words or 190 characters, the PP is released with a dayfile diagnostic - "TOO MANY CONTROL CARDS".
2. Otherwise, the job name is assembled in left-justified display code with trailing spaces. The job name may not be blank or begin with a number.
3. The priority is extracted and converted to binary. Only the lowest 4 bits are stored for the job priority.
4. The time limit is extracted and converted to binary. The lowest order 5 octal digits are rounded to the nearest $10_8$ seconds and stored for the time limit.
5. The field length is extracted and converted to binary. The lowest order 17 bits are rounded up to the nearest $100_8$ words.
6. The PPU time charges for the CP area are cleared in order to assign future PP activity to the job.
7. A dayfile message - JOB CARD ERROR - is caused by:
   a) Job name exceeding 7 characters or not beginning with an alphabetic character.
   b) Priority exceeding 7 characters.
   c) Time limit exceeding 7 characters.
   d) Field length exceeding 7 characters.
8. If any parameter is blank, a corresponding value is inserted.
   a) priority - 1
   b) time limit - $10_8$ seconds
   c) field length - $40000_8$ words

NOTES:

1. The routine READ NEXT CHARACTER reads one central memory word (10 characters) whenever the character string is depleted.

2. The parameters for the control statement buffer used by 2TJ, P60-65, are set by the circular buffer I/O routines.

3. 2TJ and the calling routine 1BJ, 1LJ, or 1LT are released and control reverts to the idle loop if one of the following conditions occur:

   a) Too many control cards - more than 190 characters in all control cards, excluding trailing blanks. About 40 cards can be used and this error usually occurs when a record separator (7-8-9 card) has been omitted.

   b) If the job name field is blank or absent.

   c) If the first character of the job name field is not an alphabetic character.

## 2TJ Routines

| Location | Routine | Calls |
|---|---|---|
| 2000 | Main Program | 2100, 2200, 2300, 2340 |
| 2100 | Assemble Argument | 2140 |
| 2140 | Read Next Character | - |
| 2200 | Decimal Conversion | 531, 12-760 |
| 2300 | Assemble Name | 2100 |
| 2340 | Clear PP Time | 04-760 |

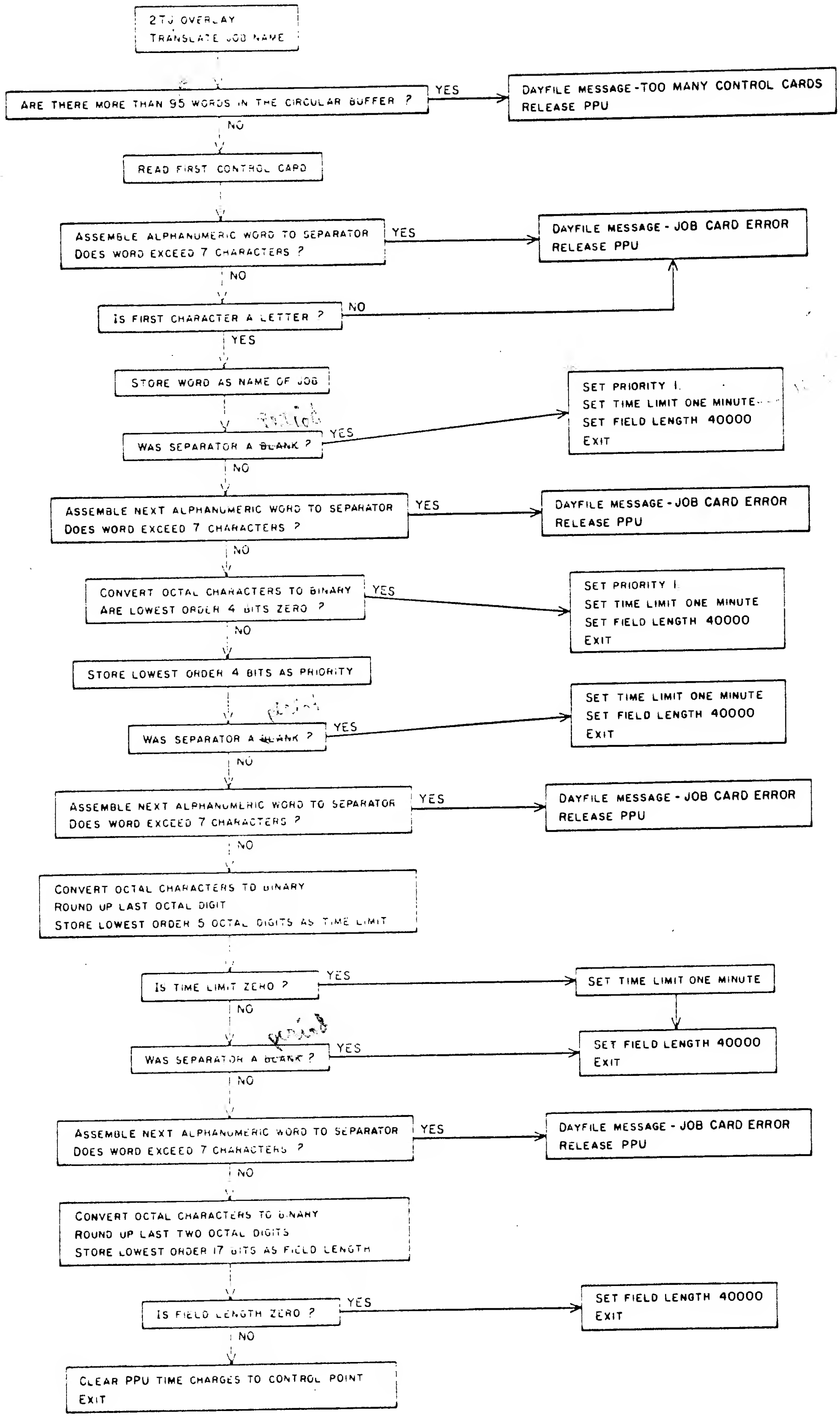
## Direct Core Cells

Entry

| | |
|---|---|
| P60/61 | FIRST |
| P62/63 | IN |
| P64/65 | OUT |
| P55 | RA |

Exit

| | |
|---|---|
| P30/34 | Job Name |
| P35 | Priority |
| P36 | Time Limit (Rounded to Tens) |
| P37 | Field Length (Rounded to Hundreds) |


2TJ OVERLAY
TRANSLATE JOB NAME
ARE THERE MORE THAN 95 WORDS IN THE CIRCULAR BUFFER ?
YES
DAYFILE MESSAGE - TOO MANY CONTROL CARDS
RELEASE PPU
NO
READ FIRST CONTROL CARD
ASSEMBLE ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
DAYFILE MESSAGE - JOB CARD ERROR
RELEASE PPU
NO
IS FIRST CHARACTER A LETTER ?
NO
YES
STORE WORD AS NAME OF JOB
WAS SEPARATOR A BLANK ?
YES
SET PRIORITY 1
SET TIME LIMIT ONE MINUTE
SET FIELD LENGTH 40000
EXIT
NO
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
DAYFILE MESSAGE - JOB CARD ERROR
RELEASE PPU
NO
CONVERT OCTAL CHARACTERS TO BINARY
ARE LOWEST ORDER 4 BITS ZERO ?
YES
SET PRIORITY 1
SET TIME LIMIT ONE MINUTE
SET FIELD LENGTH 40000
EXIT
NO
STORE LOWEST ORDER 4 BITS AS PRIORITY
WAS SEPARATOR A BLANK ?
YES
SET TIME LIMIT ONE MINUTE
SET FIELD LENGTH 40000
EXIT
NO
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
DAYFILE MESSAGE - JOB CARD ERROR
RELEASE PPU
NO
CONVERT OCTAL CHARACTERS TO BINARY
ROUND UP LAST OCTAL DIGIT
STORE LOWEST ORDER 5 OCTAL DIGITS AS TIME LIMIT
IS TIME LIMIT ZERO ?
YES
SET TIME LIMIT ONE MINUTE
NO
WAS SEPARATOR A BLANK ?
YES
SET FIELD LENGTH 40000
EXIT
NO
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
DAYFILE MESSAGE - JOB CARD ERROR
RELEASE PPU
NO
CONVERT OCTAL CHARACTERS TO BINARY
ROUND UP LAST TWO OCTAL DIGITS
STORE LOWEST ORDER 17 BITS AS FIELD LENGTH
IS FIELD LENGTH ZERO ?
YES
SET FIELD LENGTH 40000
EXIT
NO
CLEAR PPU TIME CHARGES TO CONTROL POINT
EXIT

ROUTINE: 2TS -- Translate Control Statement

PURPOSE: To examine each statement in the control card buffer of the control point area and initiate the execution.

GENERAL: This package is called by 1AJ which was in turn called by MTR to advance the job status at a control point. Each time a control statement is initiated the PP is released and MTR must then reload 1AJ. This process continues until a blank entry in the control card buffer is encountered and 1AJ can continue subsequent processing.

METHOD:

1. If the next control statement is blank, all control cards have been processed so an EXIT is made to 1AJ.

2. ASSIGN

   a) No separator is required between ASSIGN and equipment type.

   b) If either field is incorrect, an error flag of 3 is set in the control point area and an EXIT made. A "CONTROL CARD ERROR" message is sent to the dayfile and the next time 1AJ is called the PP will be aborted.

   c) The file name from the card is stored and a request is made of MTR for the octal code that the equipment type designates. If a mnemonic, i.e., WT, CR, etc., instead of octal digits, i.e., 51, 42, etc., was specified, a console message "WAITING FOR XX" will appear.

   d) The file name is assigned an FNT entry with local type status. The equipment type is set into the FST entry.

   e) The control statement buffer address is advanced so that the next statement will be processed when 1AJ is reloaded.

   f) A dayfile message noting the equipment assignment (XX ASSIGNED) is sent and the PP released.

3. COMMON

   a) If the file name exceeds 7 characters, a control card error exit is made.

   b) The FNT is searched for a file name identical to the one on the card. It must be assigned to the calling control point.

   c) If the found file is not on the disk, MTR is requested

to assign the proper equipment. If the request is not fulfilled, a console message "WAITING FOR XX" is sent and the PP released.

d) If there is no file assigned to the control point with the proper name, a console message "WAITING FOR COMMON FILE" is sent and the PP is released.

e) A file with the correct name and control point assignment is given common status and then the PP is released.

3. RELEASE

a) The FNT is searched for a common file with assignment to the requesting control point and a name identical to the control card. If one is found, the common type is changed to local so that when this job is logged off the file will be erased.

b) A dayfile message "RELEASE XXXX" is sent even if the file was not found.

c) A common file may be released by a job but still used by it because the file is not lost until the job is terminated.'

4. EXIT

a) If this control card is the next to be executed an exit is made from this overlay.

b) 1AJ checks the error flags before any control is given to 2TS. If such a flag is set, 2EF is called to read the rest of the control cards in the buffer and position the buffer parameters to the statement after an EXIT card, if one is found, or to a blank word, if no EXIT card was issued.

c) If no errors have thus far been encountered and an EXIT card found, 2TS will exit and 1AJ will finish the rest of its processing.

d) An EXIT card will cause job termination when encountered if no errors exist in the job.

5. REQUEST

a) If an equipment has not been assigned by the operator the message "REQUEST XXXX" is sent.

b) When the operator does make the assignment, the octal digits will appear in CP(22). This byte is cleared and a blank entry in the FNT is searched for.

c) The requested file will be given an FNT entry with

local type and the equipment number will be set in the FST.'

d) A dayfile message "(XX ASSIGNED)" is sent and the PP released.

6. MODE

a) The octal digit is assembled and MTR is requested to assign the corresponding exit mode.

b) A dayfile message "MODE X" is sent and the PP released.

c) MTR will change the exit mode in the exchange package for the control point.

7. SWITCH

a) FNT search

1) The FNT is searched for a file with the name identical to the one on the card and assigned to the control point. If none is found, the library is searched.

2) The file must be on disk 0.

3) The file is then read into central memory beginning at RA until an end-of-record or field length is reached.

4) The exchange area is cleared and P is then set to the number of arguments + 3 and FL is put into AO.

5) The sense switches already set in the control point are passed to RA and RA+1 is cleared.

6) The parameters on the control card are assembled and replace their corresponding entry in the argument area. Blank parameters will cause the original value to remain. A period or closing parenthesis must terminate the parameters.

7) If the RSS (read next control statement but stop before execution) flag is set, the card is sent to the dayfile and the PP is released.

8) Otherwise, the central processor is requested of MTR to begin execution of the newly loaded program and the statement is sent to the dayfile.

b) CLD search

1) Each entry in CLD is searched for the file name

and if it is found it is read into central memory beginning at RA until an end-of-record or field length is reached.

2) The program is read in from the disk in the same manner as described by (4) above.

c) PLD search

1) If the file name is not found in FNT or CLD, PLD must contain it or an error results.

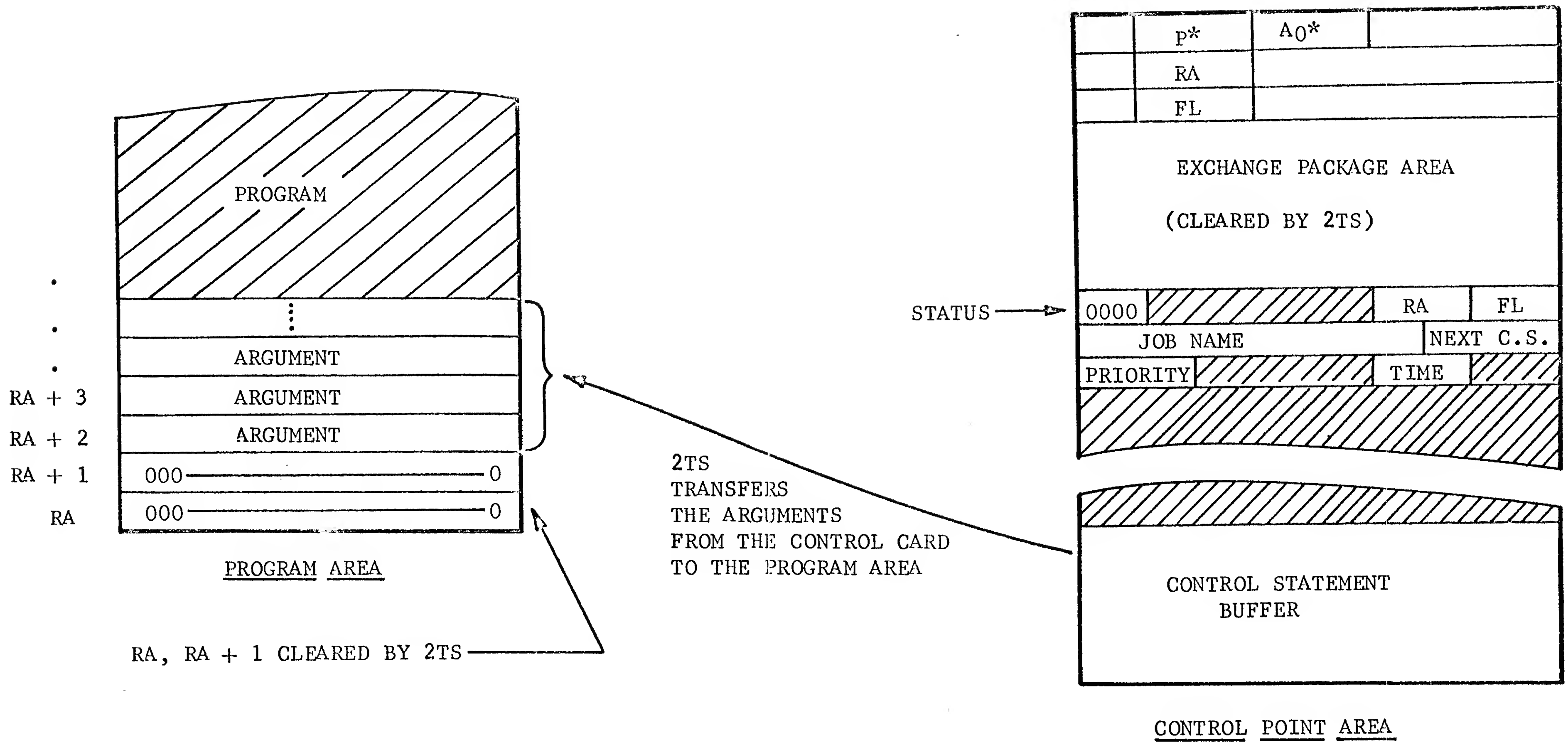
2) If the name does not begin with a letter, an error message is sent to the dayfile.

3) Parameters may appear in the call. If they do, then the first one is assembled into bits 18-35 and the second into bits 0-17 of PP recall register. If only one parameter is needed, it resides in the lowest 18 bits of the register. The call is assembled in the PP recall register at the control point so that when MTR senses this request, the package will be assigned to a free PP.

4) The statement address is advanced to the next statement and the PP is released.

## 2TS Routines

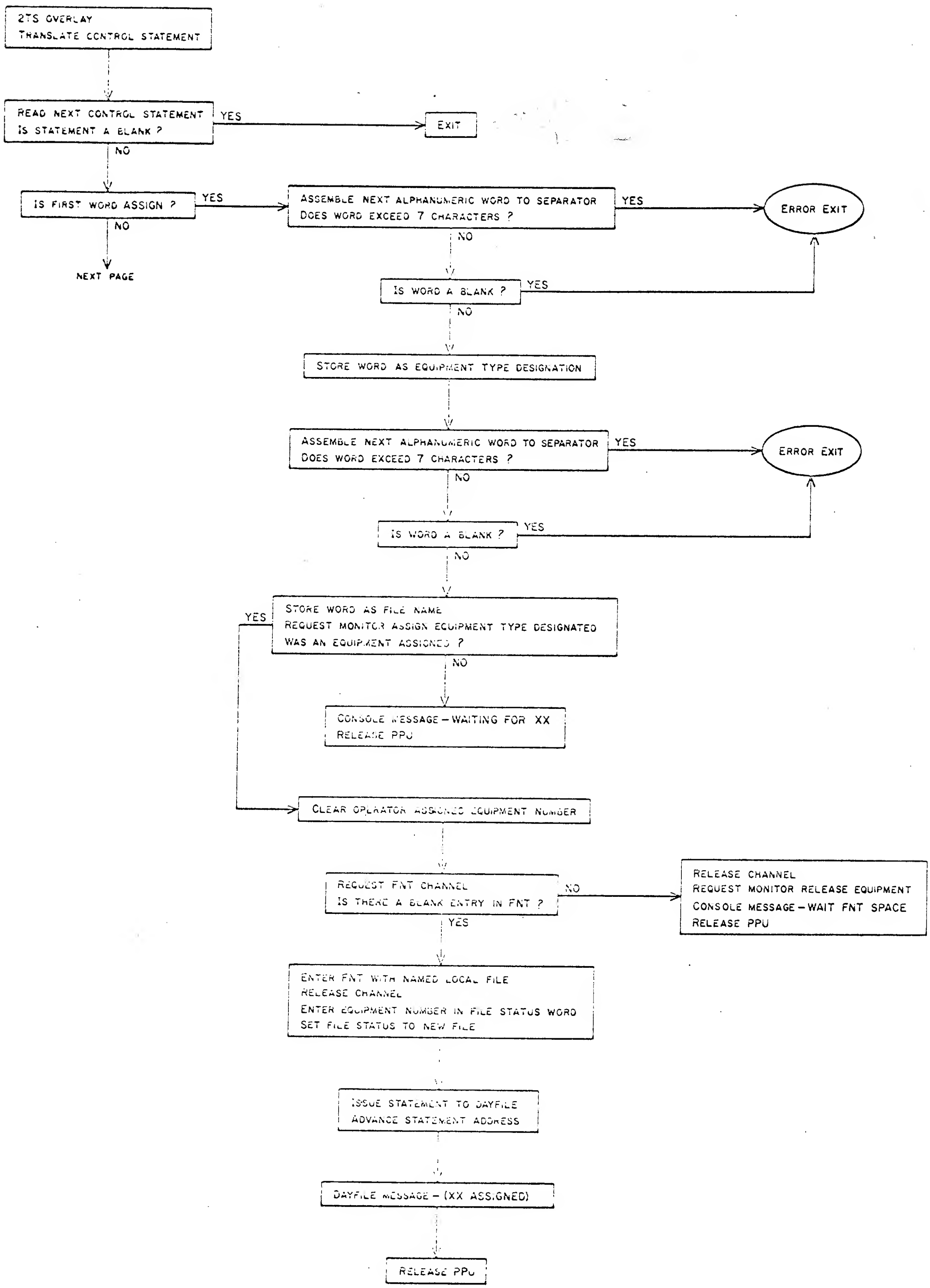
| | | |
|---|---|---|
| 2000 | Main Program | 2100, 2200, 3200, 3300, 3540, 3100 |
| 2040 | Message for ASSIGN | |
| 2100 | Unpack Next Statement | |
| 2170 | EXIT | |
| 2200 | Search for Special Format | |
| 2240 | Assemble Name | 3100 |
| 2300 | ASSIGN | 2240, 22-760, 3360, 12-760, 2500, 2040, 3100, 3000 |
| 2400 | REQUEST | 2240, 2500, 2040, 12-760, 3000 |
| 2460 | MODE | 2240, 25-760, 3000 |
| 2500 | Assign File | 740, 750, 23-760, 12-760 |
| 2600 | RELEASE | 2240, 3000 |
| 2660 | COMMON | 2240, 740, 750, 3740, 12-760, 3000 |
| 3000 | Issue Exit | 01-760, 530, 12-760 |
| 3100 | Error Exit | 3000 |
| 3150 | Enter Arguments in Program | 2240 |
| 3200 | Search for Assigned File | 2240, 3400, 3460, 3150, 15-760, 3000 |
| 3300 | Search CLD | 2240, 3400, 3460, 3150, 15-760, 3000 |
| 3360 | Console Message | |
| 3400 | Read Program | 740, 700, 400, 750 |
| 3460 | Clear Exchange Area | |
| 3540 | Search PLD | 2240, 3700, 3000 |
| 3640 | SWITCH | 2240, 3000 |
| 3700 | Assemble Data | |
| 3740 | Assign Equipment | 22-760, 3360, 750, 12-760 |



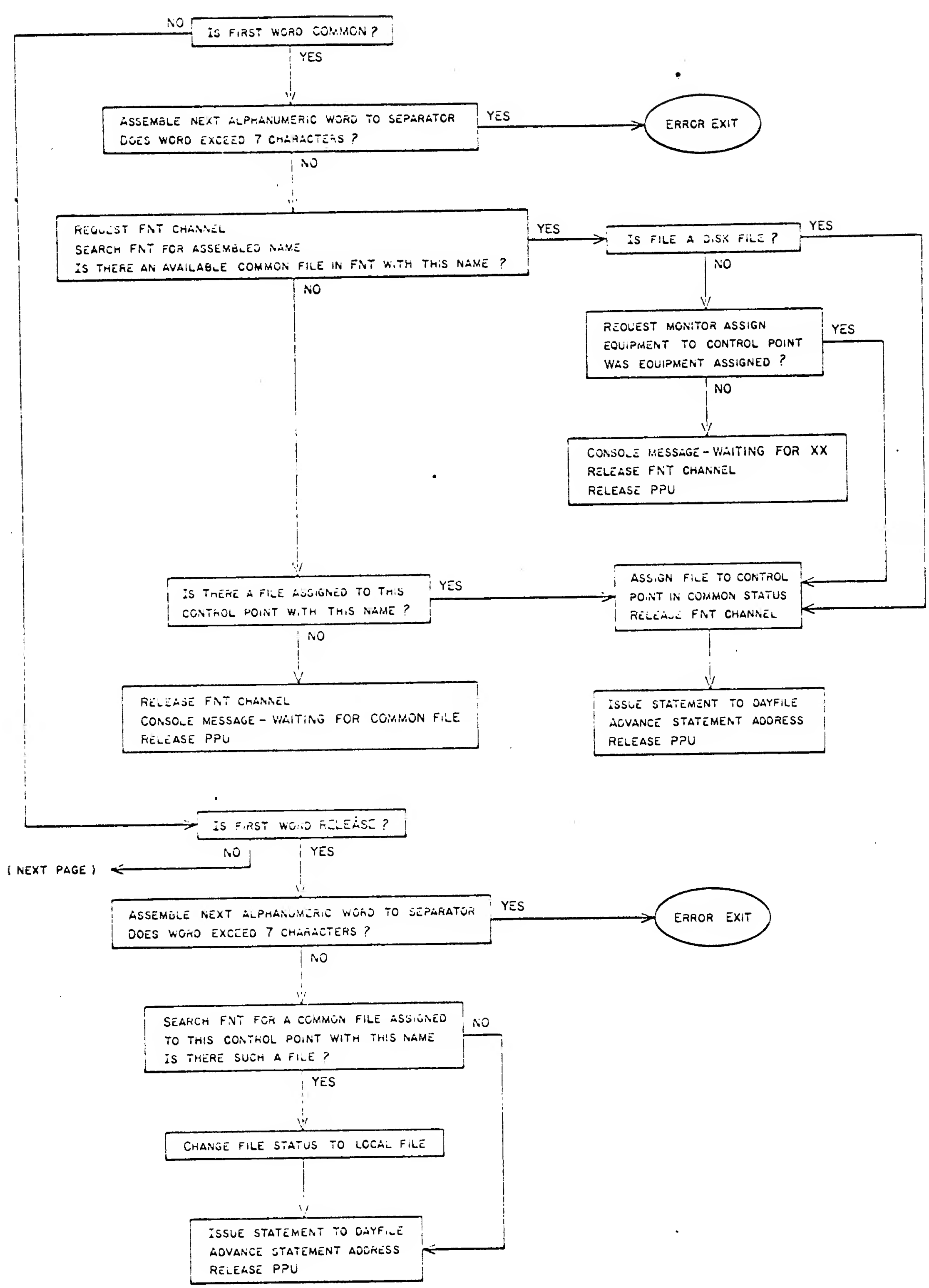

MTR DETECTS THE PRESENCE OF THE JOB AT THE CONTROL POINT AND CALLS 1AJ. 1AJ CALLS AN OVERLAY, 2TS, TO PROCESS THE NEXT CONTROL STATEMENT (ASSUMED HERE TO BE A PROGRAM CARD).

* P AND $A_0$ ARE SET BY 2TS: $A_0$ IS SET WITH THE FIELD LENGTH

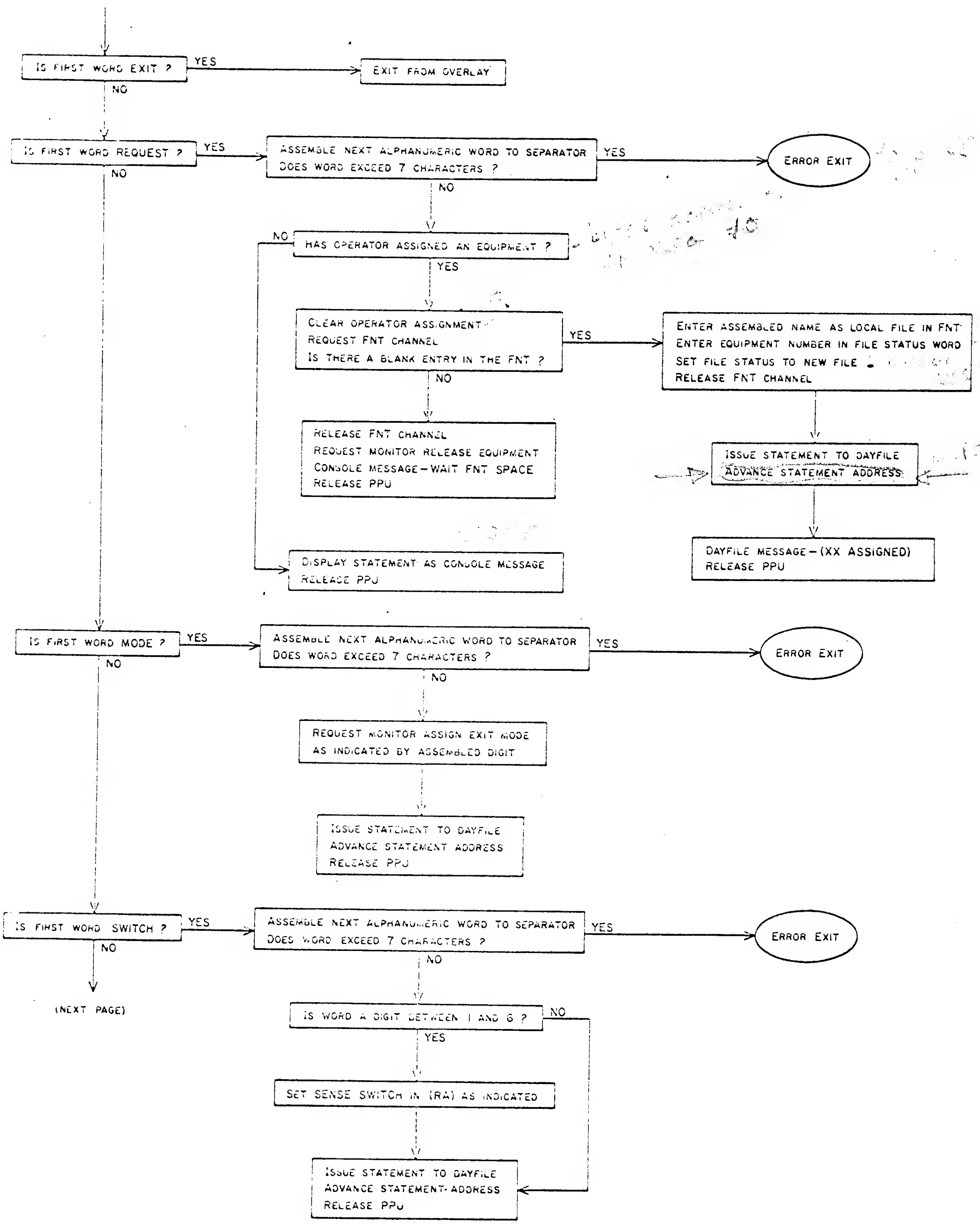
## JOB INITIATION: 1AJ/2TS

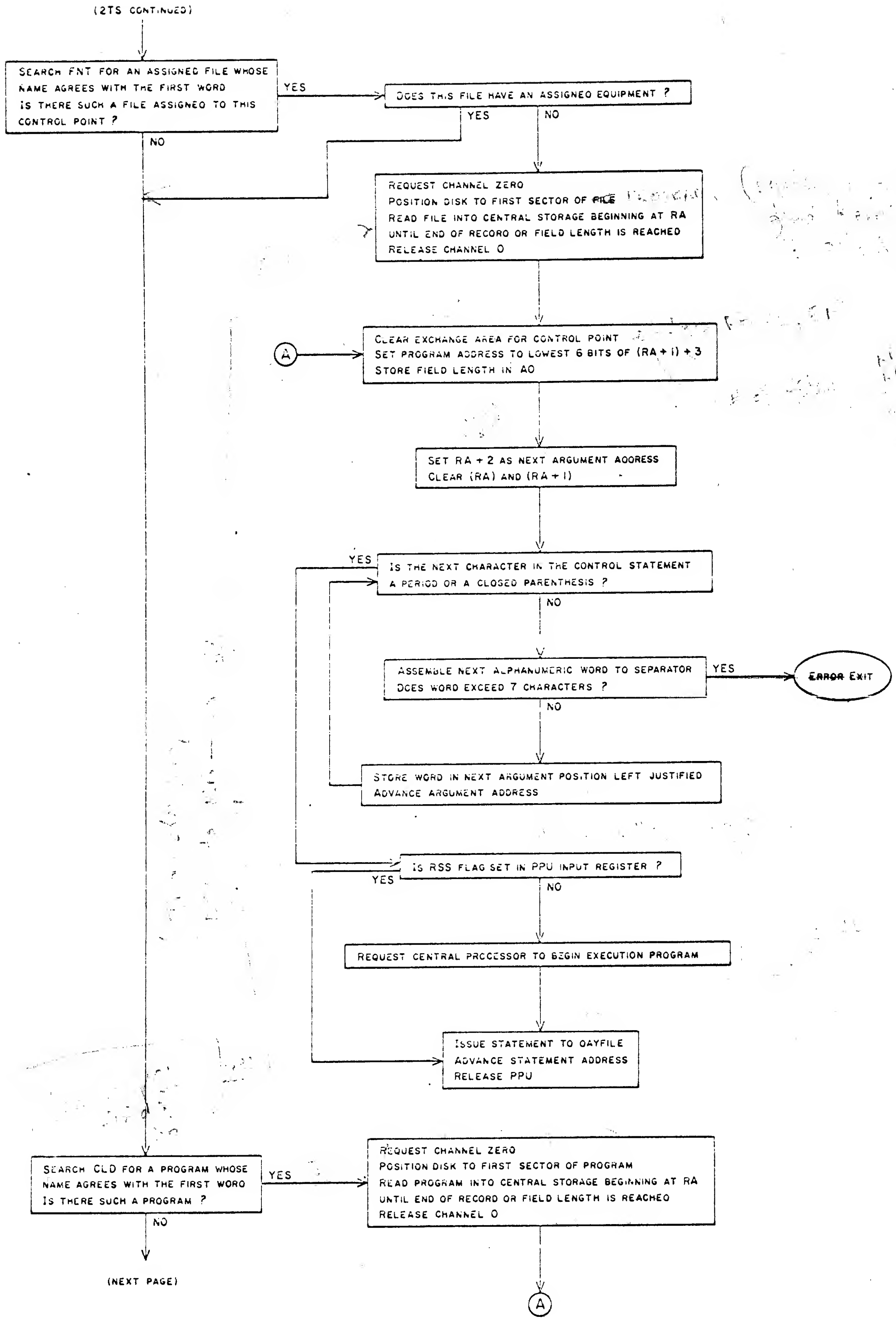

2TS OVERLAY
TRANSLATE CONTROL STATEMENT
READ NEXT CONTROL STATEMENT
IS STATEMENT A BLANK ?
YES
EXIT
NO
IS FIRST WORD ASSIGN ?
YES
NO
NEXT PAGE
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
ERROR EXIT
NO
IS WORD A BLANK ?
YES
NO
STORE WORD AS EQUIPMENT TYPE DESIGNATION
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
ERROR EXIT
NO
IS WORD A BLANK ?
YES
NO
STORE WORD AS FILE NAME
REQUEST MONITOR ASSIGN EQUIPMENT TYPE DESIGNATED
WAS AN EQUIPMENT ASSIGNED ?
YES
NO
CONSOLE MESSAGE – WAITING FOR XX
RELEASE PPU
CLEAR OPERATOR ASSIGNED EQUIPMENT NUMBER
REQUEST FNT CHANNEL
IS THERE A BLANK ENTRY IN FNT ?
NO
RELEASE CHANNEL
REQUEST MONITOR RELEASE EQUIPMENT
CONSOLE MESSAGE – WAIT FNT SPACE
RELEASE PPU
YES
ENTER FNT WITH NAMED LOCAL FILE
RELEASE CHANNEL
ENTER EQUIPMENT NUMBER IN FILE STATUS WORD
SET FILE STATUS TO NEW FILE
ISSUE STATEMENT TO DAYFILE
ADVANCE STATEMENT ADDRESS
DAYFILE MESSAGE – (XX ASSIGNED)
RELEASE PPU

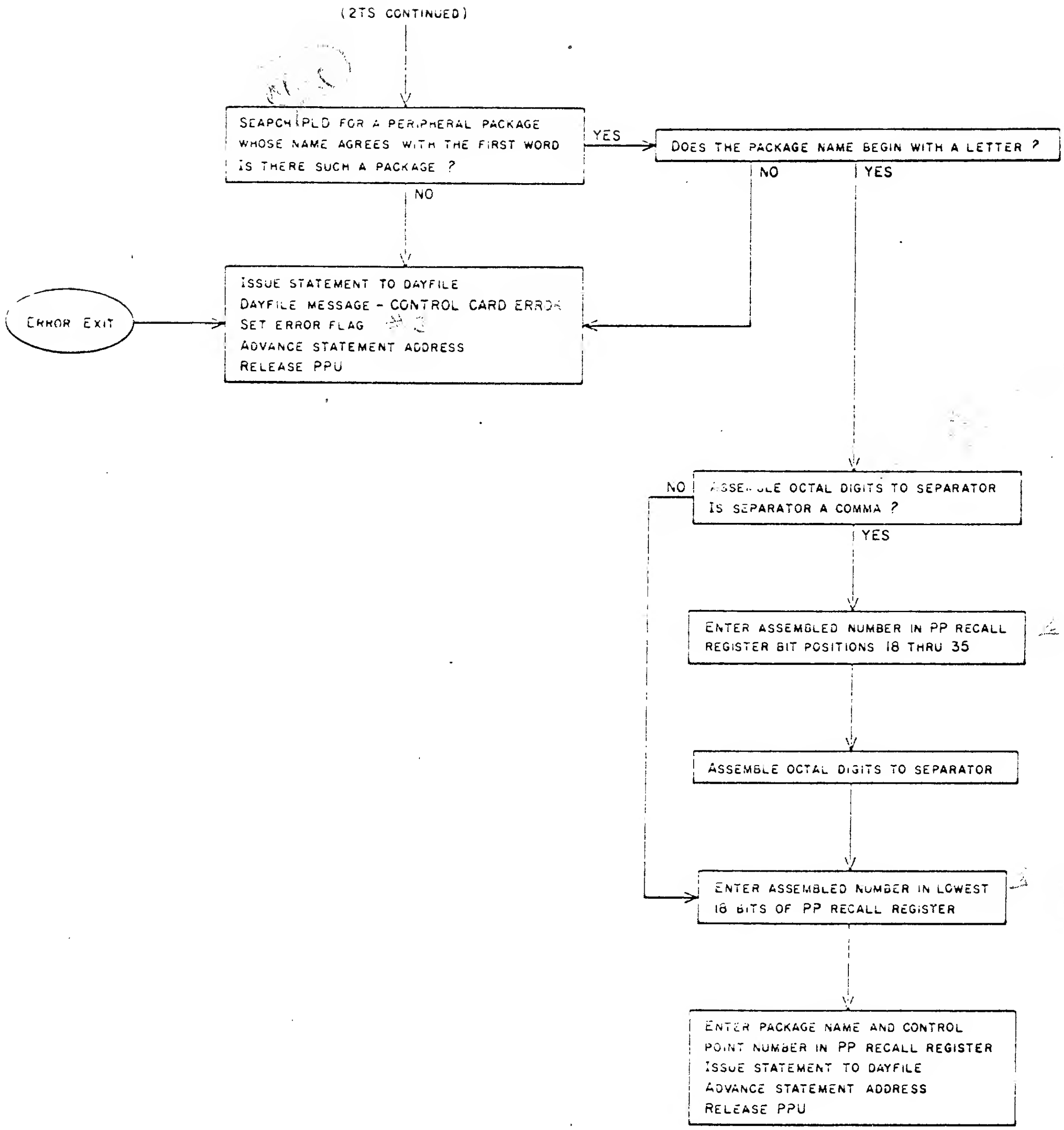
(2TS CONTINED)



(2TS CONTINUED)




(2TS CONTINUED)
SEARCH FNT FOR AN ASSIGNED FILE WHOSE NAME AGREES WITH THE FIRST WORD IS THERE SUCH A FILE ASSIGNED TO THIS CONTROL POINT ?
YES
NO
DOES THIS FILE HAVE AN ASSIGNED EQUIPMENT ?
YES
NO
REQUEST CHANNEL ZERO
POSITION DISK TO FIRST SECTOR OF FILE
READ FILE INTO CENTRAL STORAGE BEGINNING AT RA
UNTIL END OF RECORD OR FIELD LENGTH IS REACHED
RELEASE CHANNEL 0
A
CLEAR EXCHANGE AREA FOR CONTROL POINT
SET PROGRAM ADDRESS TO LOWEST 6 BITS OF (RA + 1) + 3
STORE FIELD LENGTH IN A0
SET RA + 2 AS NEXT ARGUMENT ADDRESS
CLEAR (RA) AND (RA + 1)
YES
IS THE NEXT CHARACTER IN THE CONTROL STATEMENT A PERIOD OR A CLOSED PARENTHESIS ?
NO
ASSEMBLE NEXT ALPHANUMERIC WORD TO SEPARATOR
DOES WORD EXCEED 7 CHARACTERS ?
YES
ERROR EXIT
NO
STORE WORD IN NEXT ARGUMENT POSITION LEFT JUSTIFIED
ADVANCE ARGUMENT ADDRESS
IS RSS FLAG SET IN PPU INPUT REGISTER ?
YES
NO
REQUEST CENTRAL PROCESSOR TO BEGIN EXECUTION PROGRAM
ISSUE STATEMENT TO DAYFILE
ADVANCE STATEMENT ADDRESS
RELEASE PPU
SEARCH CLD FOR A PROGRAM WHOSE NAME AGREES WITH THE FIRST WORD
IS THERE SUCH A PROGRAM ?
YES
NO
REQUEST CHANNEL ZERO
POSITION DISK TO FIRST SECTOR OF PROGRAM
READ PROGRAM INTO CENTRAL STORAGE BEGINNING AT RA
UNTIL END OF RECORD OR FIELD LENGTH IS REACHED
RELEASE CHANNEL 0
A
(NEXT PAGE)


(2TS CONTINUED)
SEARCH PLD FOR A PERIPHERAL PACKAGE WHOSE NAME AGREES WITH THE FIRST WORD
IS THERE SUCH A PACKAGE ?
YES
NO
DOES THE PACKAGE NAME BEGIN WITH A LETTER ?
NO
YES
ERROR EXIT
ISSUE STATEMENT TO DAYFILE
DAYFILE MESSAGE - CONTROL CARD ERROR
SET ERROR FLAG
ADVANCE STATEMENT ADDRESS
RELEASE PPU
ASSEMBLE OCTAL DIGITS TO SEPARATOR
IS SEPARATOR A COMMA ?
NO
YES
ENTER ASSEMBLED NUMBER IN PP RECALL REGISTER BIT POSITIONS 18 THRU 35
ASSEMBLE OCTAL DIGITS TO SEPARATOR
ENTER ASSEMBLED NUMBER IN LOWEST 18 BITS OF PP RECALL REGISTER
ENTER PACKAGE NAME AND CONTROL POINT NUMBER IN PP RECALL REGISTER
ISSUE STATEMENT TO DAYFILE
ADVANCE STATEMENT ADDRESS
RELEASE PPU

ROUTINE: 2WT -- Write Tape

PURPOSE: To write both binary and BCD blocks of data on magnetic tape.

GENERAL: Once a write code is detected in the request parameter, a call is made to the CIO Write Function routine which then checks the equipment type of the file. When a file type of tape is determined, a call is made to load 2WT. When the mode of binary or BCD is determined, the appropriate transfer is made by CIO.

METHOD: A. BINARY WRITE

1. The circular buffer is checked to determine if there is a full block of data. If there is not, and an end-of-record function is not requested, execution returns to the CIO Write Function routine.

2. If end-of-record is requested, the last partial record will be written.

3. A transfer is made to subroutine Write Binary Tape in 2WT to actually write the block. A check is made for tape ready. If the tape is not ready, a message is displayed and a pause is executed waiting for tape to be made ready or the error flag set in RA.

4. Once tape is ready, the data is written and a parity check is made. If there is parity, a message is displayed, the tape is backspaced, and rewritten until either the parity does not exist or the error flag has been set in RA by monitor.

5. If a good write is performed, the OUT address of the buffer is then updated. If a short block was written meaning end-of-buffer, IN and OUT are set equal to FIRST and EXIT is taken to CIO Write Function.

6. If the buffer is not empty, more data is written until a short record is encountered.

B. BCD WRITE

1. If the request is a BCD write request, a jump is made from CIO Write Function to the subroutine WRITE BCD TAPE at location 2640.

2. A check is made to see if there is data in the buffer. If there is none, and the end-of-record is requested, IN and OUT are set equal FIRST. If end-of-record is not requested, an EXIT is taken.

3. If the buffer is not empty, one word at a time is read from the buffer and it is converted from display code to BCD advancing OUT as each word is read. Whenever the last byte of a word is zero, the line is padded with spaces up to 120 characters.

4. When a full line of data is made up, a jump to 3001 is taken (WRITE CODED RECORD) to write the record. The same write and parity checking operation is done here as in the binary write.

5. When a good write is completed the block count is advanced, the channel released, and more data is written until the buffer is empty.

C. WRITE FILE MARK

1. If a file mark is requested, a jump is taken from CIO Write Function to WRITE FILE MARK.

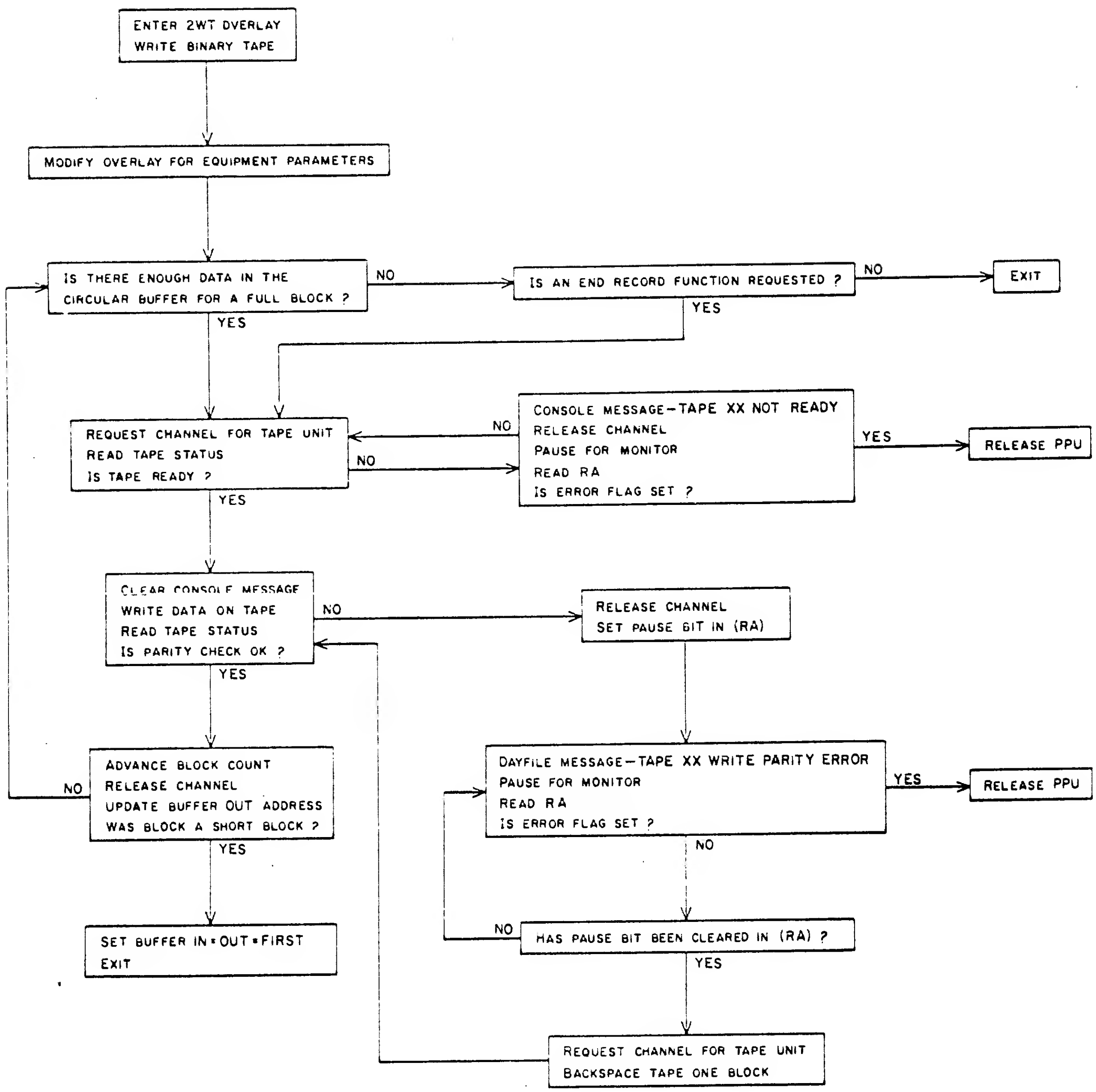
2. This routine simply finds the tape, makes sure it is ready, writes a file mark, advances the block count, and releases the channel.

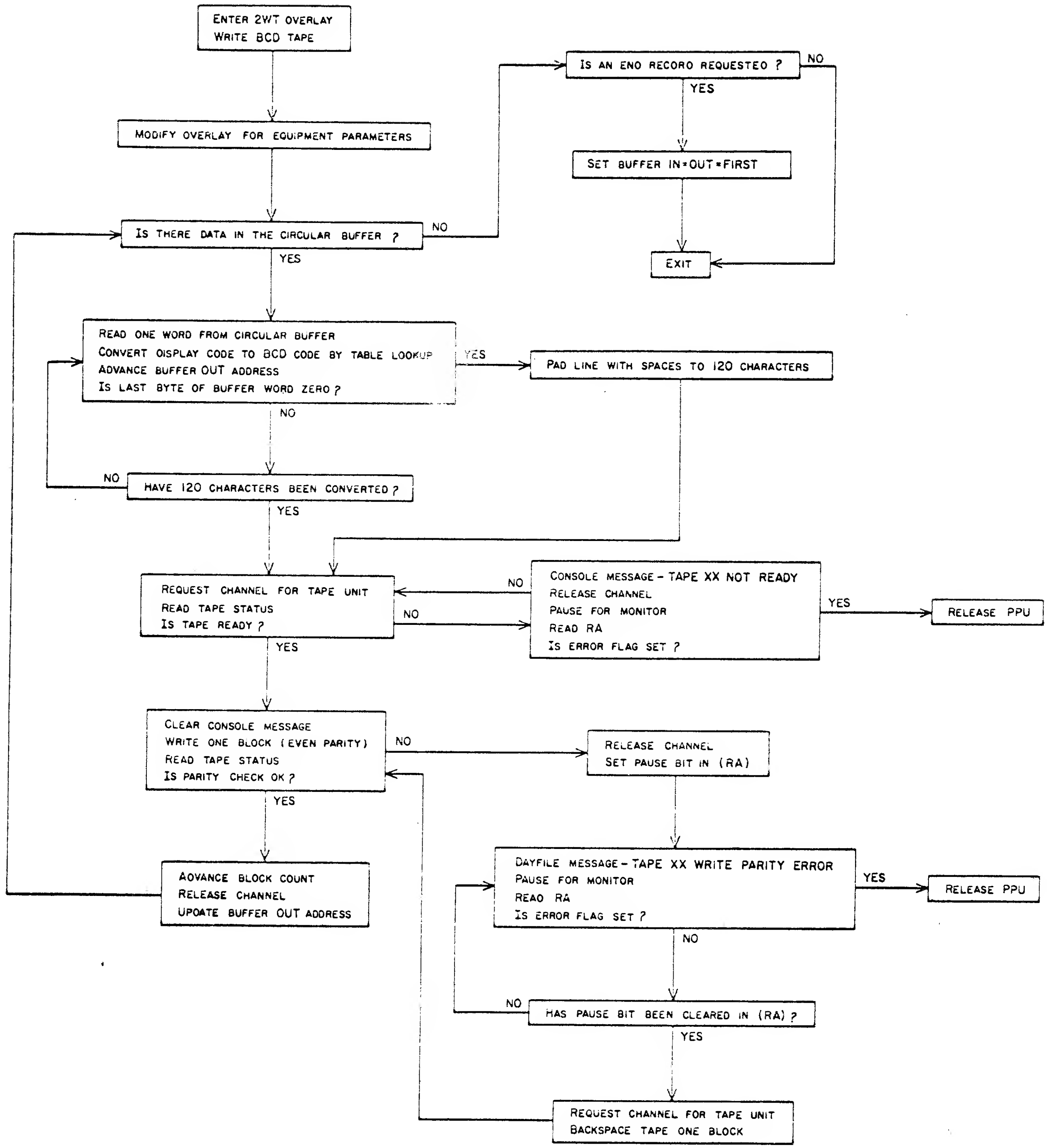
NOTES:

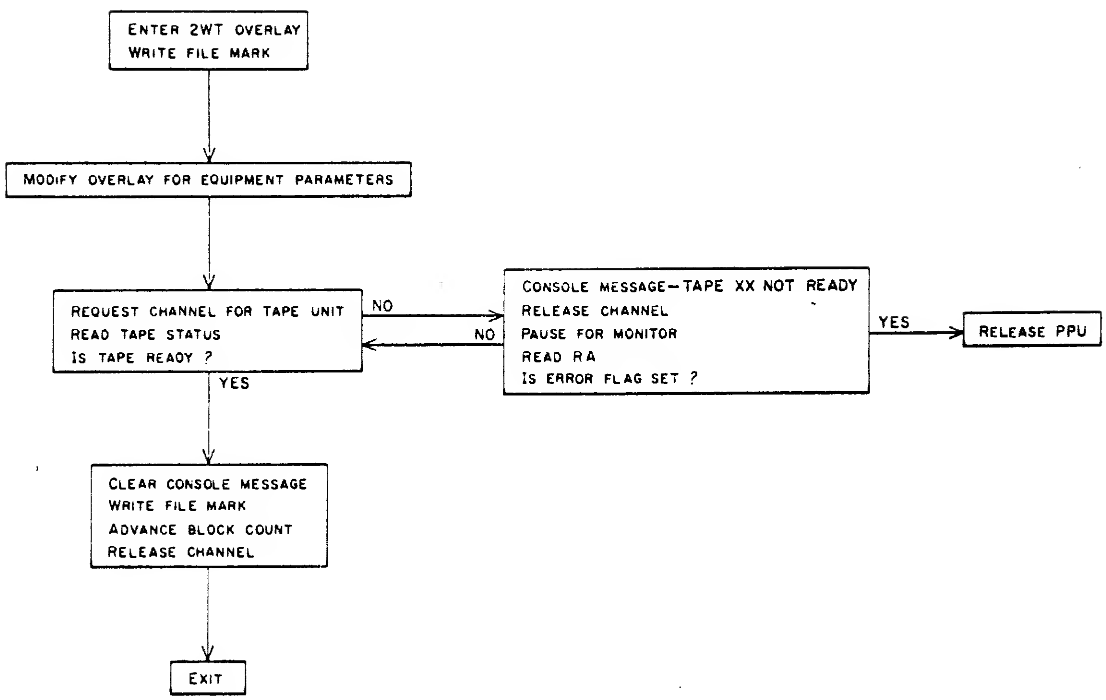
1. An end-or-record write must be issued to empty a buffer which does not contain a full block of data.

2. Binary tape records has a maximum size of $1000_8$ central memory words.

3. BCD tape records are all 120 characters (one print line). Each record is padded with spaces to maintain the proper size.


ENTER 2WT OVERLAY
WRITE BINARY TAPE
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
IS THERE ENOUGH DATA IN THE CIRCULAR BUFFER FOR A FULL BLOCK ?
NO
IS AN END RECORD FUNCTION REQUESTED ?
NO
EXIT
YES
YES
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
NO
CONSOLE MESSAGE—TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
YES
RELEASE PPU
YES
CLEAR CONSOLE MESSAGE
WRITE DATA ON TAPE
READ TAPE STATUS
IS PARITY CHECK OK ?
NO
RELEASE CHANNEL
SET PAUSE BIT IN (RA)
YES
ADVANCE BLOCK COUNT
RELEASE CHANNEL
UPDATE BUFFER OUT ADDRESS
WAS BLOCK A SHORT BLOCK ?
NO
YES
SET BUFFER IN=OUT=FIRST
EXIT
DAYFILE MESSAGE—TAPE XX WRITE PARITY ERROR
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
HAS PAUSE BIT BEEN CLEARED IN (RA) ?
YES
REQUEST CHANNEL FOR TAPE UNIT
BACKSPACE TAPE ONE BLOCK


ENTER 2WT OVERLAY
WRITE BCD TAPE
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
IS THERE DATA IN THE CIRCULAR BUFFER ?
NO
IS AN END RECORD REQUESTED ?
NO
YES
SET BUFFER IN=OUT=FIRST
EXIT
YES
READ ONE WORD FROM CIRCULAR BUFFER
CONVERT DISPLAY CODE TO BCD CODE BY TABLE LOOKUP
ADVANCE BUFFER OUT ADDRESS
IS LAST BYTE OF BUFFER WORD ZERO ?
YES
PAD LINE WITH SPACES TO 120 CHARACTERS
NO
NO
HAVE 120 CHARACTERS BEEN CONVERTED ?
YES
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
NO
NO
CONSOLE MESSAGE - TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
YES
CLEAR CONSOLE MESSAGE
WRITE ONE BLOCK (EVEN PARITY)
READ TAPE STATUS
IS PARITY CHECK OK ?
NO
RELEASE CHANNEL
SET PAUSE BIT IN (RA)
YES
ADVANCE BLOCK COUNT
RELEASE CHANNEL
UPDATE BUFFER OUT ADDRESS
DAYFILE MESSAGE - TAPE XX WRITE PARITY ERROR
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
HAS PAUSE BIT BEEN CLEARED IN (RA) ?
YES
REQUEST CHANNEL FOR TAPE UNIT
BACKSPACE TAPE ONE BLOCK


ENTER 2WT OVERLAY
WRITE FILE MARK
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
REQUEST CHANNEL FOR TAPE UNIT
READ TAPE STATUS
IS TAPE READY ?
NO
CONSOLE MESSAGE—TAPE XX NOT READY
RELEASE CHANNEL
PAUSE FOR MONITOR
READ RA
IS ERROR FLAG SET ?
NO
YES
RELEASE PPU
YES
CLEAR CONSOLE MESSAGE
WRITE FILE MARK
ADVANCE BLOCK COUNT
RELEASE CHANNEL
EXIT

## Line Limit

The seventh argument on the RUN card for the Chippewa FORTRAN compiler is an octal line limit. This limit applies only to the standard output file ("OUTPUT") of an object program. If not specified on the RUN card, the line limit is set by the compiler to $10,000_8$. If, during the execution of the object program, the number of lines written to the output file exceeds the line limit, the job is aborted. The line limit is checked by the execution time output routine OUTPUTC.

CONTROL DATA CORPORATION

Development Division - Applications

ALPHABETIC PERIPHERAL PACKAGES

Chippewa Operating System

10/20/65

Table of Contents

# ALPHABETIC PERIPHERAL PACKAGES

## INTRODUCTION

The packages described on the following pages may be called by a central program. They are loaded into a peripheral processor from either RPL (resident peripheral library) or PLD (peripheral library directory). A central program, by setting the package name in left-justified display code in RA+1, requests MTR to assign the package to a free PP. Each package begins execution at location 1000 in the PP and arguments are passed to it from the central program through the lower portion of RA+1. If the execution of the package is terminated normally or abnormally the PP is released and must be reassigned when it is needed again.

The last section of this narrative gives a few practical examples about the use of some of the routines.

ROUTINE: DMP -- Storage Dump

PURPOSE: To enter an octal dump of a requested area of central memory into the OUTPUT file.

GENERAL: This package may be called by a control card or DIS console. Three calls may be made:

a) no parameters - dump only exchange package

b) one argument - dump from RA to the specified address

c) two arguments - dump area between the two addresses.

METHOD:

1. Two checks made on the arguments passed through the input register may cause a diagnostic:

   a) terminal address < initial address

   b) terminal address > field length

   Either condition will cause a "DMP ARG ERROR" dayfile message and the control point aborted.

2. In the case that both parameters are equal, i.e., usually zero, the exchange jump area (first 16 words of control point area) is set up as the dump address. The title of the dump is changed to "DMPX."

3. The FNT is searched for a file of local or common and assigned to this control point. The name must be OUTPUT and the file on disk 0 with buffer status indicating not busy (odd value). If no such file is found, an entry of this type is made into FNT so that the dump can be printed. The file status in either case is set to $14_8$ (request coded write).

4. If no OUTPUT file was found while searching FNT, then the new file just added must have a track assignment. A track is requested of MTR and when it is assigned the number is inserted in the FST entry of the new file.

5. If the last reference to the file was a read operation, then no dumping will be done. This prevents writing over output data that may have been repositioned by the read.

6. The dump has a header of either DMPX, for an exchange area dump, or DMP. for any other dump. Each central memory word has an address relative to RA and 4 five digit groups of data with two spaces between the address and data and a space separating each byte. The peripheral buffer spans from 2000-7000 and is filled before it is passed to the output file.

7. The dump address is incremented by one until the terminal address specified in the input register is reached. A return jump is issued to dump the PP buffer into the OUTPUT file when it is full or the terminal address is encountered.

8. The PP buffer is written on disk 0 a full sector ($100_8$ words) at a time until short sector is found. It is written on the disk followed by a file mark and then channel 0 is released. The buffer input address is reset so that more data may be inserted if the terminal address has not been reached. Every write to the disk is terminated by a file mark but the sector number is not incremented. This will prevent a file from ever running away but still allow more information to erase the file mark and reside within one file.

9. After the formatted octal dump has been successfully passed to the OUTPUT file, the buffer status byte in FST is changed to $15_8$ (completed coded write). Then the PP is released.

NOTES:

1. Successive identical lines are not suppressed.

2. One print line contains only an address and a central memory word.

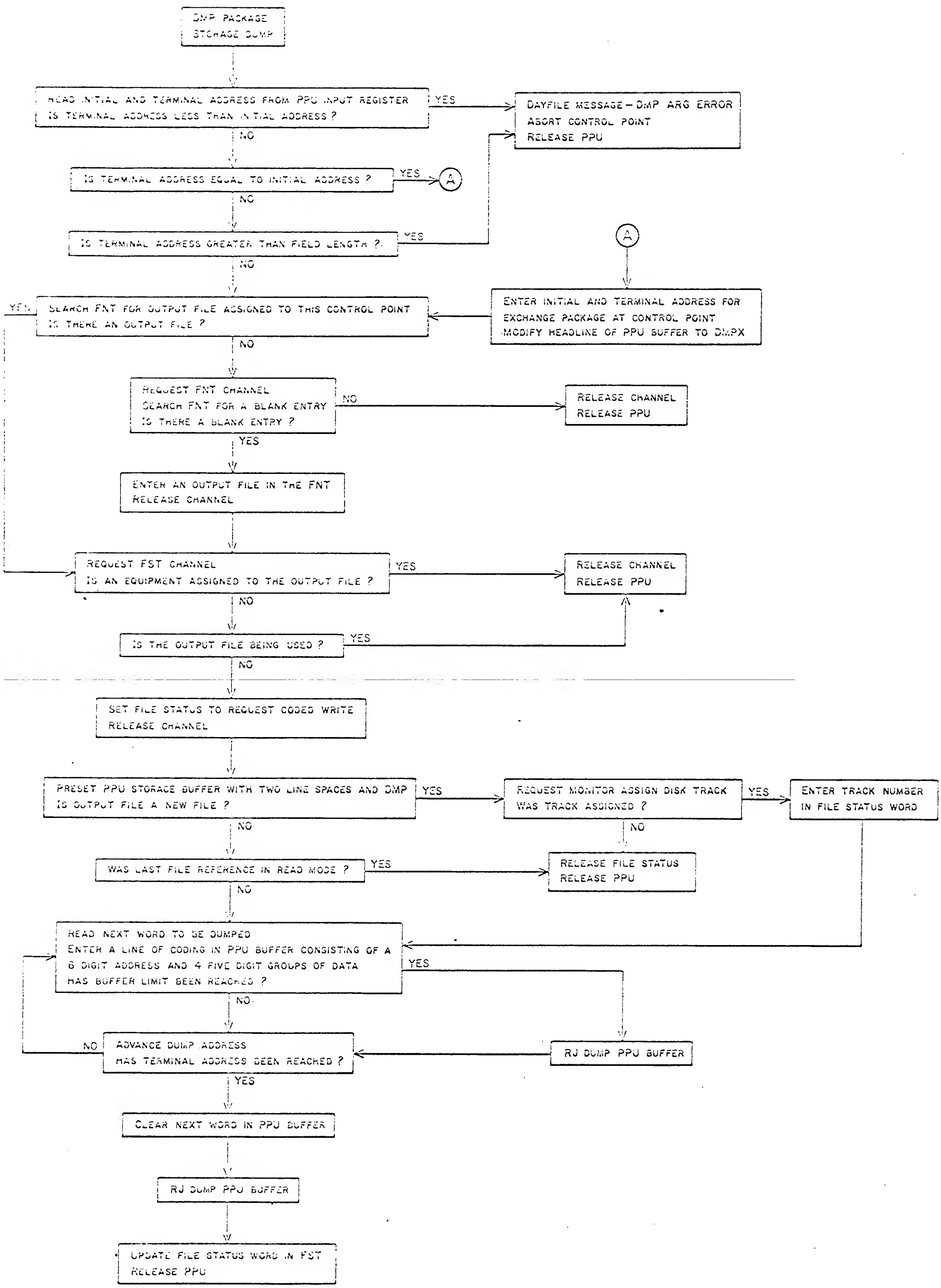
## DMP Routines

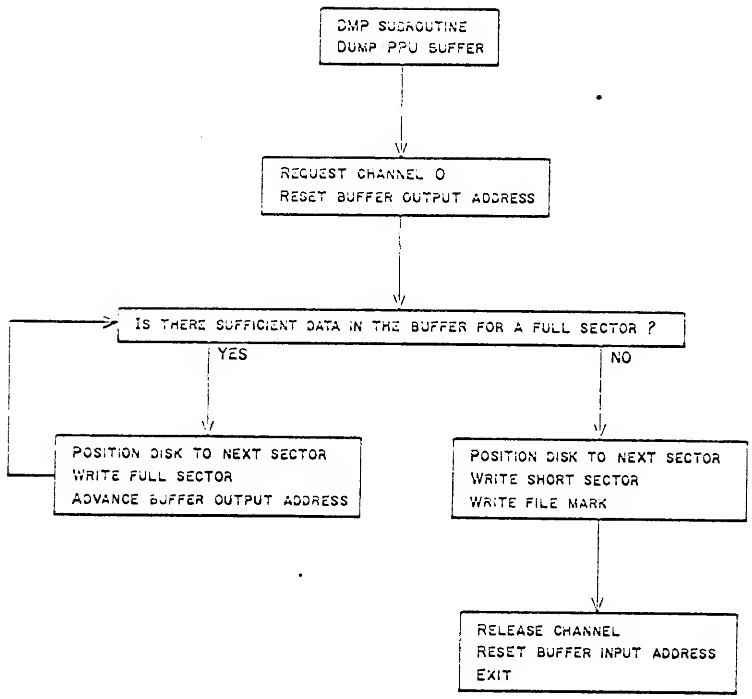
| | | |
|---|---|---|
| 1000 | Main Program | 1560, 1100, 1200, 12-760, 100, 530, 13-760 |
| 1100 | Search for Output File | 740, 750, 12-760, 100 |
| 1200 | Enter Output File | 6-760, 1300, 1600 |
| 1300 | Enter Line in Buffer | 1600 |
| 1560 | Process Exchange Area | |
| 1600 | Dump Buffer | 740, 1700, 700, 1740, 750 |
| 1700 | Enter Control Byte | 6-760 |
| 1740 | Write Sector | |
| 2000 | Disk Buffer | |
| 2030 | Begin Output File | 15-740, 750, 12-760, 100 |

## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P10/14 | CP Status |
| | P50/54 | Input register contents |
| | P55 | RA |
| | P60/61 | First argument |
| | P62/63 | Second argument |
| | P74 | CP address |
| | P75 | Input register address |
| 1100 | P01 | File type (local or common) |
| | P10/14 | FNT entry |
| | P20/24 | FNT status later FST entry |
| | P45 | Last buffer status from FST |
| | P50/54 | Input register contents |
| | P57 | FST address |
| 1200 | P01 | Central memory word count |
| | P10/14 | PP message buffer contents |
| | P20/24 | FST entry |
| | P45 | Last buffer status from FST |
| | P57 | FST address |
| | P60/61 | First argument |

| | | |
|---|---|---|
| | P62/63 | Second argument |
| | P64 | IN address for PP buffer |
| 1300 | P10/14 | Central memory word to be dumped |
| | P60/61 | First argument |
| | P64 | IN address for PP buffer |
| 1560 | P60/61 | First argument |
| | P62/63 | Second argument |
| | P55 | RA |
| | P74 | CP address |
| 1600 | P01 | Central memory word count |
| | P02 | Sector length |
| | P20/24 | FST entry |
| | P64 | IN address for PP buffer |
| | P65 | OUT address for PP buffer |
| 1700 | P02 | Disk status byte |
| | P10/14 | Message buffer |
| | P20/24 | FST entry |
| 1740 | P01 | Disk status byte from FST |
| | P20/24 | FST entry |
| 2030 | P01 | FNT index |
| | P10/14 | FNT entry |
| | P20/24 | FNT status |


DMP PACKAGE
STORAGE DUMP
READ INITIAL AND TERMINAL ADDRESS FROM PPU INPUT REGISTER
IS TERMINAL ADDRESS LESS THAN INITIAL ADDRESS ?
YES
DAYFILE MESSAGE – DMP ARG ERROR
ABORT CONTROL POINT
RELEASE PPU
NO
IS TERMINAL ADDRESS EQUAL TO INITIAL ADDRESS ?
YES
A
NO
IS TERMINAL ADDRESS GREATER THAN FIELD LENGTH ?
YES
NO
A
ENTER INITIAL AND TERMINAL ADDRESS FOR
EXCHANGE PACKAGE AT CONTROL POINT
MODIFY HEADLINE OF PPU BUFFER TO DMPX
YES
SEARCH FNT FOR OUTPUT FILE ASSIGNED TO THIS CONTROL POINT
IS THERE AN OUTPUT FILE ?
NO
REQUEST FNT CHANNEL
SEARCH FNT FOR A BLANK ENTRY
IS THERE A BLANK ENTRY ?
NO
RELEASE CHANNEL
RELEASE PPU
YES
ENTER AN OUTPUT FILE IN THE FNT
RELEASE CHANNEL
REQUEST FST CHANNEL
IS AN EQUIPMENT ASSIGNED TO THE OUTPUT FILE ?
YES
RELEASE CHANNEL
RELEASE PPU
NO
IS THE OUTPUT FILE BEING USED ?
YES
NO
SET FILE STATUS TO REQUEST CODED WRITE
RELEASE CHANNEL
PRESET PPU STORAGE BUFFER WITH TWO LINE SPACES AND DMP
IS OUTPUT FILE A NEW FILE ?
YES
REQUEST MONITOR ASSIGN DISK TRACK
WAS TRACK ASSIGNED ?
YES
ENTER TRACK NUMBER
IN FILE STATUS WORD
NO
RELEASE FILE STATUS
RELEASE PPU
NO
WAS LAST FILE REFERENCE IN READ MODE ?
YES
NO
READ NEXT WORD TO BE DUMPED
ENTER A LINE OF CODING IN PPU BUFFER CONSISTING OF A
6 DIGIT ADDRESS AND 4 FIVE DIGIT GROUPS OF DATA
HAS BUFFER LIMIT BEEN REACHED ?
YES
NO
NO
ADVANCE DUMP ADDRESS
HAS TERMINAL ADDRESS BEEN REACHED ?
RJ DUMP PPU BUFFER
YES
CLEAR NEXT WORD IN PPU BUFFER
RJ DUMP PPU BUFFER
UPDATE FILE STATUS WORD IN FST
RELEASE PPU


DMP SUBROUTINE
DUMP PPU BUFFER
REQUEST CHANNEL 0
RESET BUFFER OUTPUT ADDRESS
IS THERE SUFFICIENT DATA IN THE BUFFER FOR A FULL SECTOR ?
YES
NO
POSITION DISK TO NEXT SECTOR
WRITE FULL SECTOR
ADVANCE BUFFER OUTPUT ADDRESS
POSITION DISK TO NEXT SECTOR
WRITE SHORT SECTOR
WRITE FILE MARK
RELEASE CHANNEL
RESET BUFFER INPUT ADDRESS
EXIT

ROUTINE EXU - Execute Compiled Program

PURPOSE To locate and read a specified file from the disk into central memory. The appropriate exchange jump package parameters are set up and then the central processor is told that the file is ready for execution.

GENERAL After a file has been compiled and stored on the disk, EXU is used to load a file into central memory beginning at the calling program's reference address. The location of the name of the file (left-justified display code) to be called and executed is set in the lower 18 bits of the input register.

METHOD 1. The error flag at the control point is checked. If it is set, the package is released so that error processing may proceed.

2. The file name is read in by adding RA and the lower 18 bits of the input register. FNT is searched for the file name and if it is located a check is made on its control point assignment.

3. When the file is located, its type from the FNT is checked for input and output. Only common or local files may be executed.

4. The FST entry must reflect that the file is on disk and has been used.

5. A dayfile message of "PROGRAM NOT ON DISK" is sent if:

   a) The file name was not located in the FNT.
   b) The file was not assigned to the calling control point.
   c) The file has either an input or output status.
   d) The file has an equipment other than disk assigned, i.e. it is a card file or tape file.
   e) The file has not been used, i.e. no track has been assigned. This status is reflected by checking the beginning track byte in the FST for non-zero.

6. A request for channel 0 is made and the disk is positioned to the beginning track and sector for the file.

7. The file is read and stored one sector at a time into central memory beginning at the control points' reference address. Encountering a short sector or reaching the field limit causes the reading of the file to be terminated.

8. If the field limit was reached before the end of the file, a dayfile message of "PROGRAM TOO LONG" appears and the control point aborted.

9. The exchange jump package in the control point area is updated to permit execution of the newly loaded program.

a) First the sense lights and switches from word 26 of the control point are stored in RA.

b) RA+1 is read and then cleared.

c) P in the echange jump area is set to the number of parameters from RA+1 plus 3. The field length (in hundreds) from word 20 is stored in A0.

d) RA and FL remain the same values, but all of the other registers are cleared.

10. The central processor is then requested by a MTR code $15_8$. When this request has been processed, the PP is released.

NOTES

1. The calling program is completely overlayed by the file read in off the disk.

2. Sense lights and switches are passed from the calling program to the new program through RA.

3. The field length specified in RA of the called program is ignored. Only the field limit assigned initially to the control point is checked.

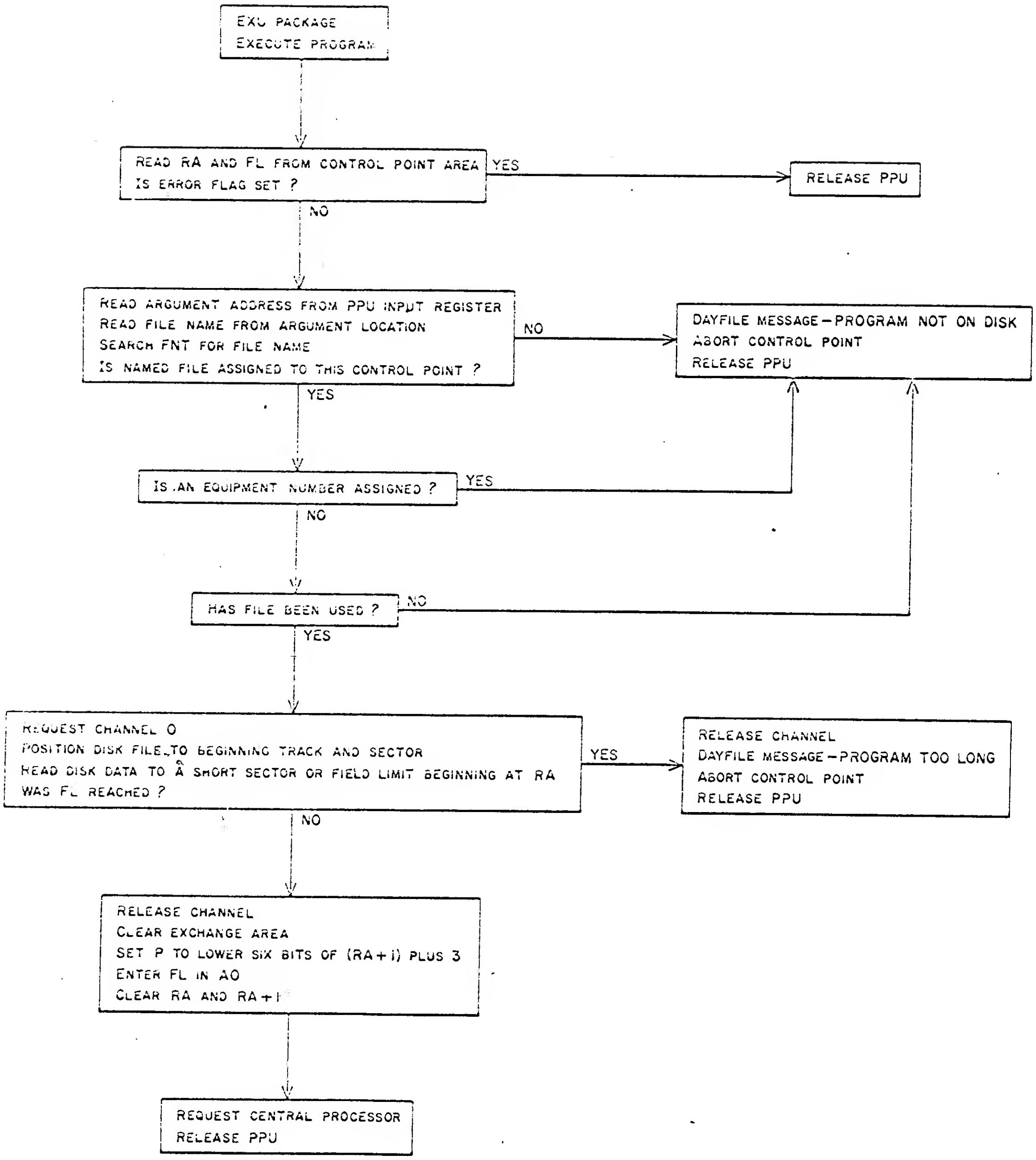
## EXU Routines

| | | |
|---|---|---|
| .1000 | Main Program | 12-760, 100, 1100, 1200, 1300, 15-760, 531, 13-760 |
| 1100 | Search for file | 1160, 1064 |
| 1200 | Read program from disk | 740, 700, 400, 750 |
| 1300 | Clear exchange area | |

## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P06 | beginning track number of file |
| | P07 | sector number |
| | P10/14 | CP(20) - status word |
| | P20/24 | contents of FST entry |
| | P50/54 | contents of input register |
| | P55 | RA |
| | P56 | FL |
| | P57 | FST status |
| | P74 | control point address |
| | P75 | address of input register |
| | P7200/7702 | disk buffer |
| 1100 | P01 | control point assignment |
| | P10/14 | FNT entry, later FST entry |
| | P20/24 | File name in left-justified display code |
| | P30/34 | FNT status |

| | | |
|---|---|---|
| 1200 | P01 | control point assignment |
| | P04 | RA (in hundreds) |
| | P05 | FL (in hundreds) |
| | P06 | track number |
| | P07 | sector number |
| 1300 | P01 | control point assignment |
| | P10/14 | zeroed, later each word of exchange area |
| | P20/24 | CP (26), later RA+1 |
| | P30/34 | CP (20) |


EXU PACKAGE
EXECUTE PROGRAM
READ RA AND FL FROM CONTROL POINT AREA
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
READ ARGUMENT ADDRESS FROM PPU INPUT REGISTER
READ FILE NAME FROM ARGUMENT LOCATION
SEARCH FNT FOR FILE NAME
IS NAMED FILE ASSIGNED TO THIS CONTROL POINT ?
NO
DAYFILE MESSAGE—PROGRAM NOT ON DISK
ABORT CONTROL POINT
RELEASE PPU
YES
IS AN EQUIPMENT NUMBER ASSIGNED ?
YES
NO
HAS FILE BEEN USED ?
NO
YES
REQUEST CHANNEL 0
POSITION DISK FILE TO BEGINNING TRACK AND SECTOR
READ DISK DATA TO A SHORT SECTOR OR FIELD LIMIT BEGINNING AT RA
WAS FL REACHED ?
YES
RELEASE CHANNEL
DAYFILE MESSAGE—PROGRAM TOO LONG
ABORT CONTROL POINT
RELEASE PPU
NO
RELEASE CHANNEL
CLEAR EXCHANGE AREA
SET P TO LOWER SIX BITS OF (RA+1) PLUS 3
ENTER FL IN A0
CLEAR RA AND RA+1
REQUEST CENTRAL PROCESSOR
RELEASE PPU

ROUTINE: CLL -- Central Library Loader

PURPOSE: To load one or more overlays into an area specified by a central memory calling program.

GENERAL: The location (BA) of the overlay parameters is set into the lower 18 bits of the input register. The location (BA)

| | | |
|---|---|---|
| BA | | FWA |
| BA+1 | | LIMIT |
| BA+2 | NAME | ADDR |
| | ⋮ | |
| BA+n+2 | 0-------------------- | 0 |

FWA - beginning address for first overlay

LIMIT - last address for group of overlays

NAME - name of overlay (left-justified display code)

The address of where the overlay begins will be returned in the lower 18 bits of its location in the BA area (ADDR). If it cannot be loaded because the length exceeds LIMIT, an address of $777777_8$ will be inserted. The address will remain cleared if the overlay cannot be located. A zero word must terminate the parameters.

METHOD:

1. The RA and FL are read from CP(20) and stored in hundreds.

2. From the input register the location of BA is read and incremented by RA so the first parameter is read.

3. When the LIMIT is read, a check is made to insure that it is within the field length. If LIMIT exceeds FL, the PP is released and no diagnostic results.

4. Each argument is read and checked for zero. If it is zero, then the list is assumed to be exhausted.

5. The resident subroutine library (RSL) is first searched. The first entry in RSL is checked against the name of the overlay. If a match is not found, then the field length of the RSL entry is added to the beginning address of RSL, in order to find the next subroutine in the table.

6. If the overlay is found in RSL, the length is added to FWA and that total may not exceed LIMIT. If it does, then $77777_8$ is entered into the beginning address area of that overlay (ADDR). The next argument will be read and processed.

7. FWA will reflect the next available location for loading so it is stored as ADDR for that argument. The program is transferred $100_8$ words at a time and FWA is increased by the number of words stored until a short record is encountered. A zero length record is not transmitted.

8. FWA is increased to the next available central memory address and then the next argument is processed.

9. If the name is not found in RSL then the central library directory (CLD) is searched, The format of this table is:

| NAME (DISPLAY CODE) | SEC | TRACK |
|---|---|---|
| 42 | 6 | 12 |

It is terminated by a zero word or table limit.

10. When the overlay is found to be in CLD, then FWA is stored as the overlay's beginning address. Channel 0 for the disk is requested and it is positioned to the proper track.

11. One sector at a time is read and its length recorded. If a zero length is found, it is not transmitted. If the sector length exceeds the number of words to LIMIT then they are not stored and $77777_8$ is set as the beginning address. Track repositioning is checked after every read. If the short sector is encountered before the LIMIT exceeded, it is stored and FWA is updated to be the next available program address. Another argument is then processed.

12. If the name was not found in the RSL or CLD, then the job file is read. If the package is found in the FNT, then it must be assigned to the calling program's control point and be on disk 0. If it is not, then the next argument is processed.

13. When a file is found in the FNT, the same disk operations apply as those with CLD.

14. When an argument is found to be zero then the next available program address (FWA) is zeroed. BA is also cleared to inform the calling program that CLL was finished. Then an MTR code of $15_8$, requesting the control processor is made and the PP released.

NOTES:

1. The FORTRAN compiler uses CLL to load its subroutines.

2. All files loaded by CLL are compiled to execute from 0. Therefore, if a program wanted to take advantage of this feature, all K portions of the instructions must be modified for a different starting point.

3. The last overlay loaded by CLL is followed by a zero word.

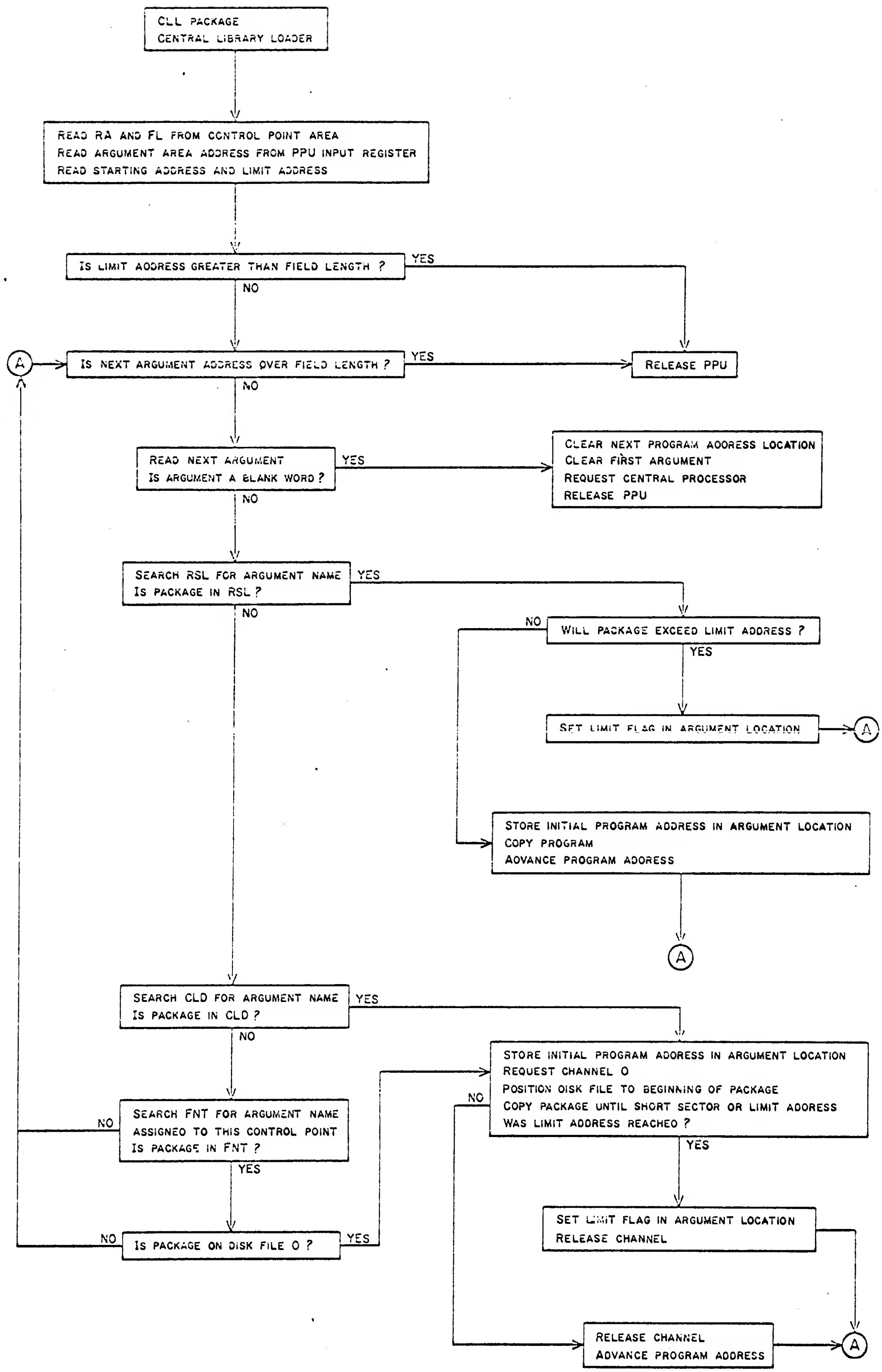
## CLL Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1101, 1201, 15-761, 12-761,100 |
| 1100 | Read arguments | 12-761 |
| 1200 | Process argument | 1301, 1501, 1601 |
| 1300 | Search RSL | |
| 1500 | Search CLD | 1601 |
| 1600 | Enter program from disk | 741, 701, 401, 751 |

## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P10/14 | CP address |
| | P20/24 | Argument |
| | P50/54 | Contents of Input register |
| | P55 | RA in hundreds |
| | P56 | FL |
| | P57 | Constant 100 |
| | P60/61 | FWA - next available program address |
| | P74 | CP address |
| | P75 | Address of Input register |
| 1100 | P10/14 | FWA and later LIMIT |
| | P50/54 | Contents of Input register |
| | P55 | RA |
| | P56 | FL |
| | P60/61 | Location of BA |
| | P62/63 | FWA |
| | P64/65 | LIMIT |
| 1200 | P01 | CP assignment |
| | P06 | Track number |
| | P07 | Sector number |
| | P10/14 | FNT entry |
| | P20/24 | Contents of Input register |
| | P30/34 | FNT status |
| | P74 | Address of Input register |

| | | |
|---|---|---|
| 1300 | P01 | Number of words read |
| | P10/14 | RSL entry |
| | P14 | Total number of words transferred |
| | P20/24 | Contents of Input register |
| | P30/34 | RSL status |
| | P55 | RA |
| | P57 | Constant 100 |
| | P62/63 | FWA, next available program address |
| | P64/65 | LIMIT |
| | P7200/7302 | Input buffer |
| 1500 | P06 | Track number |
| | P07 | Sector number |
| | P10/14 | CLD entry |
| | P20/24 | Input register |
| | P30/34 | CLD status |
| 1600 | P01 | Sector length |
| | P04/05 | Number of words to LIMIT |
| | P06 | Track number |
| | P07 | Sector number |
| | P20/24 | Input register |
| | P55 | RA |
| | P60/61 | BA |
| | P62/63 | FWA |
| | P64/65 | LIMIT |


CLL PACKAGE
CENTRAL LIBRARY LOADER
READ RA AND FL FROM CONTROL POINT AREA
READ ARGUMENT AREA ADDRESS FROM PPU INPUT REGISTER
READ STARTING ADDRESS AND LIMIT ADDRESS
IS LIMIT ADDRESS GREATER THAN FIELD LENGTH ?
YES
NO
IS NEXT ARGUMENT ADDRESS OVER FIELD LENGTH ?
YES
RELEASE PPU
A
NO
READ NEXT ARGUMENT
IS ARGUMENT A BLANK WORD ?
YES
CLEAR NEXT PROGRAM ADDRESS LOCATION
CLEAR FIRST ARGUMENT
REQUEST CENTRAL PROCESSOR
RELEASE PPU
NO
SEARCH RSL FOR ARGUMENT NAME
IS PACKAGE IN RSL ?
YES
NO
WILL PACKAGE EXCEED LIMIT ADDRESS ?
NO
YES
SET LIMIT FLAG IN ARGUMENT LOCATION
A
STORE INITIAL PROGRAM ADDRESS IN ARGUMENT LOCATION
COPY PROGRAM
ADVANCE PROGRAM ADDRESS
A
SEARCH CLD FOR ARGUMENT NAME
IS PACKAGE IN CLD ?
YES
NO
STORE INITIAL PROGRAM ADDRESS IN ARGUMENT LOCATION
REQUEST CHANNEL 0
POSITION DISK FILE TO BEGINNING OF PACKAGE
COPY PACKAGE UNTIL SHORT SECTOR OR LIMIT ADDRESS
WAS LIMIT ADDRESS REACHED ?
NO
YES
SEARCH FNT FOR ARGUMENT NAME
ASSIGNED TO THIS CONTROL POINT
IS PACKAGE IN FNT ?
NO
YES
SET LIMIT FLAG IN ARGUMENT LOCATION
RELEASE CHANNEL
IS PACKAGE ON DISK FILE 0 ?
NO
YES
RELEASE CHANNEL
ADVANCE PROGRAM ADDRESS
A

ROUTINE: LBC -- Loading Binary Corrections

PURPOSE: To load binary cards from the INPUT file into central memory.

GENERAL: The lower 18 bits of the input register contain a beginning address for the card loading. If the address is zero, the binary cards are loaded beginning at RA. It may be called via a control card or from a DIS console.

METHOD:

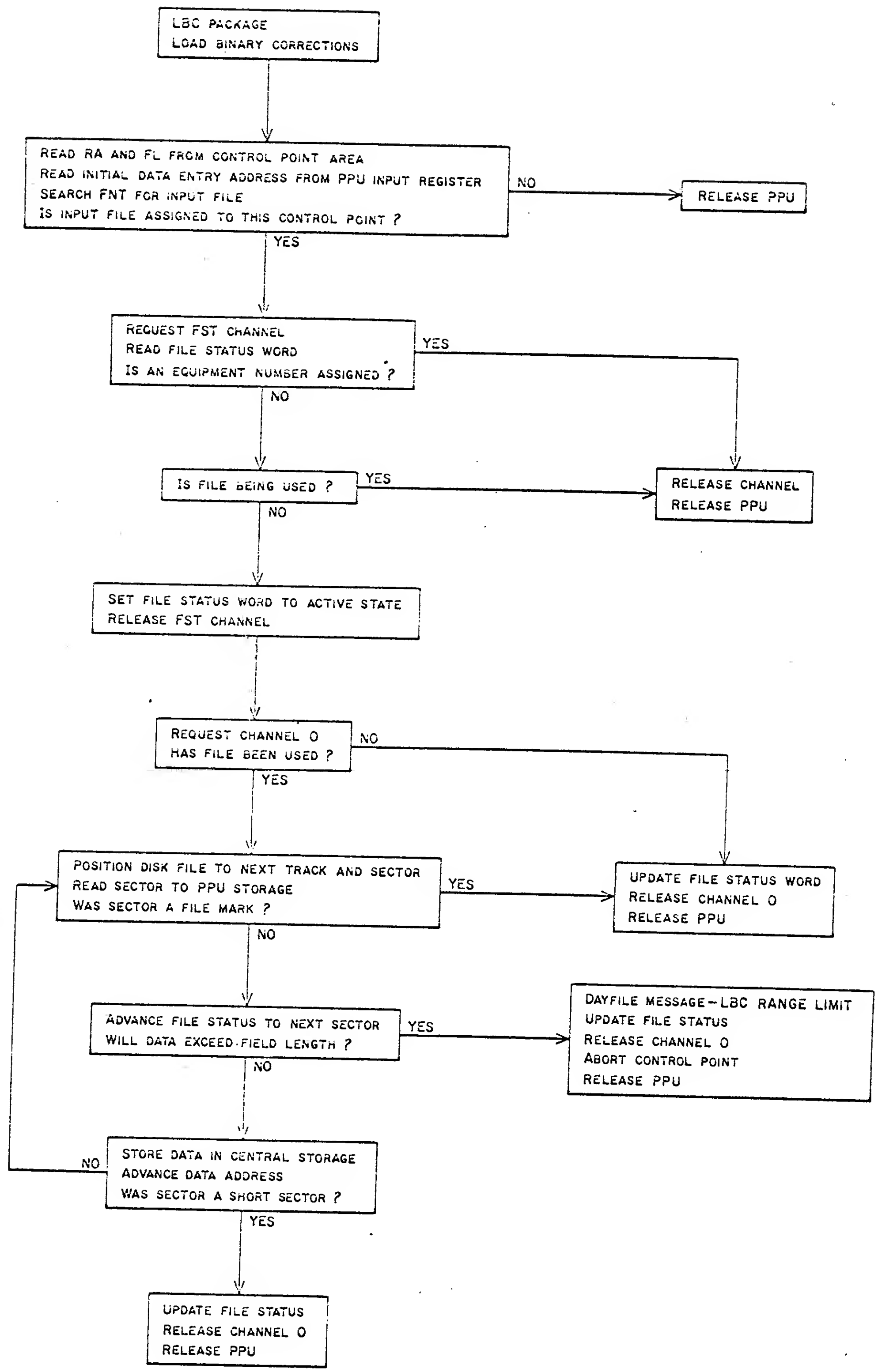
1. From the control point status word (20), the RA and FL are read.
2. Each entry in the FNT is searched for type local and assignment to this control point.
3. If no entry is found, then the PPU is released without a diagnostic.
4. When an entry is found, the file name is checked against INPUT. If it does not match, the search of FNT continues.
5. After the INPUT file is located, the FST entry is checked. The file must be on disk 0 or the PPU is released.
6. The last buffer status is checked. If it is even, then the file is being used and no action will be taken. If it is odd, then the file has no operation begin performed on it, so the status is decreased by one to make it active. When another PP wants to access this file, the buffer status will reflect an even number informing the requesting PP that the file is being acted upon.
7. The disk is positioned to the track stated in the FST and one sector is read into PP memory. After each read, a check is made for file mark and data exceeding field length. If a file mark is encountered, the buffer status is made odd and the PP released. A dayfile message "LBC RANGE LIMIT" appears if the field length would be exceeded thereby also causing the buffer status to be changed and the CP aborted.
8. After the sector read is checked, it is transferred to central memory at the location specified from the input register.
9. Only one record will be read from the INPUT file so when a short sector is encountered, the buffer status is changed and the PP is released.

## LBC Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1200, 740, 700, 400, 750, 12-760, 530, 13-760 |
| 1200 | Search for Input file | 740, 12-760, 750 |

## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P01 | Sector length |
| | P06 | Track number |
| | P07 | Sector number |
| | P10/14 | CP status (word 20) |
| | P20/24 | FST entry |
| | P50/54 | Contents of input register |
| | P55 | RA (in hundreds) |
| | P56 | FL |
| | P74 | CP address |
| | P75 | Address of input register |
| | P7200/7702 | Disk buffer |
| 1200 | P01 | File type of local and CP |
| | P10/14 | FNT entry |
| | P20/24 | FNT status |
| | P20/24 | FST entry |
| | P50/54 | Contents of input register |
| | P57 | FST address |


LBC PACKAGE
LOAD BINARY CORRECTIONS
READ RA AND FL FROM CONTROL POINT AREA
READ INITIAL DATA ENTRY ADDRESS FROM PPU INPUT REGISTER
SEARCH FNT FOR INPUT FILE
IS INPUT FILE ASSIGNED TO THIS CONTROL POINT ?
NO
RELEASE PPU
YES
REQUEST FST CHANNEL
READ FILE STATUS WORD
IS AN EQUIPMENT NUMBER ASSIGNED ?
YES
NO
IS FILE BEING USED ?
YES
RELEASE CHANNEL
RELEASE PPU
NO
SET FILE STATUS WORD TO ACTIVE STATE
RELEASE FST CHANNEL
REQUEST CHANNEL 0
HAS FILE BEEN USED ?
NO
YES
POSITION DISK FILE TO NEXT TRACK AND SECTOR
READ SECTOR TO PPU STORAGE
WAS SECTOR A FILE MARK ?
YES
UPDATE FILE STATUS WORD
RELEASE CHANNEL 0
RELEASE PPU
NO
ADVANCE FILE STATUS TO NEXT SECTOR
WILL DATA EXCEED FIELD LENGTH ?
YES
DAYFILE MESSAGE—LBC RANGE LIMIT
UPDATE FILE STATUS
RELEASE CHANNEL 0
ABORT CONTROL POINT
RELEASE PPU
NO
STORE DATA IN CENTRAL STORAGE
ADVANCE DATA ADDRESS
WAS SECTOR A SHORT SECTOR ?
NO
YES
UPDATE FILE STATUS
RELEASE CHANNEL 0
RELEASE PPU

ROUTINE: LOC -- Load Octal Corrections

PURPOSE: To make octal corrections to a program already residing in central memory.

GENERAL: Three calls may be made to this package:

a) A call without parameters will change the central memory words specified on cards in the next INPUT record.

b) With one parameter, central memory is cleared from RA to the address specified and the cards in the next INPUT record are assembled.

c) Two parameters cause the memory between the two arguments to be cleared and then the correction cards to be read.

METHOD:

1. RA and FL are read from CP (word 20).

2. The arguments, beginning address and terminal address, of where the corrections are to be inserted are checked.

   a) First greater than second.

   b) Second greater than field length.

3. If the two arguments are not equal, the central memory contained within the two is cleared.

4. The FNT is searched for a file INPUT associated with this control point and of local or common type. If one is not found, no diagnostic results, but the PP is released.

5. The proper file must be on disk file 0 and have an odd buffer status (not busy). If either condition is not met, then the PP is released.

6. By decreasing the buffer status by one, this control point puts the file in active status.

7. Channel 0 is requested for the disk which is positioned to the proper track from the FST. The PPU buffer is filled with the octal correction cards from INPUT until the buffer is either full or a short sector is encountered.

8. The cards have trailing spaces suppressed by a zero byte and are written in $100_8$ word sectors. Since the buffer is $5000_8$ PP words long, many sectors may be read. Each sector has a two word control byte which is not useful data to the program. In order to have all the useful data packed, the last two words of the previous sector are temporarily stored out of the buffer and the next sector is read over their initial location. When the

control bytes have been used, the two words are restored to their buffer positions and the last two words of the sector just read are temporarily stored out of the buffer.

9. When the buffer is either filled or all the correction cards read, INPUT is put into an inactive state (status is odd) so that another PP may use it.

10. Each octal correction card is unpacked into a character string buffer (one character per word). A zero byte terminates the unpacking of one card.

11. When the line buffer is loaded, the address is assembled. The address must be between column 1 and column 7. Spaces are suppressed and leading zeroes are not necessary. If a non-octal digit appears, the address is not assembled and no diagnostic is given.

12. After the address is assembled, the data word is packed. The data must begin after column 7 and contain 20 digits. If a non-octal digit appears the word is not assembled and no diagnostic is given.

13. The assembled address is checked against field length and is not inserted into its position if it exceeds FL. The assembled word is then entered into its assembled address.

NOTES:

1. If corrections are to be made to a binary deck, LBC (load binary cards) should be used before LOC. LOC only makes changes to programs already in central memory.

2. Central memory may be cleared using LOC only if an empty record appears in the INPUT file.

3. On the correction cards, the address must end before column 7. Spacing is not important and leading zeroes may be dropped.

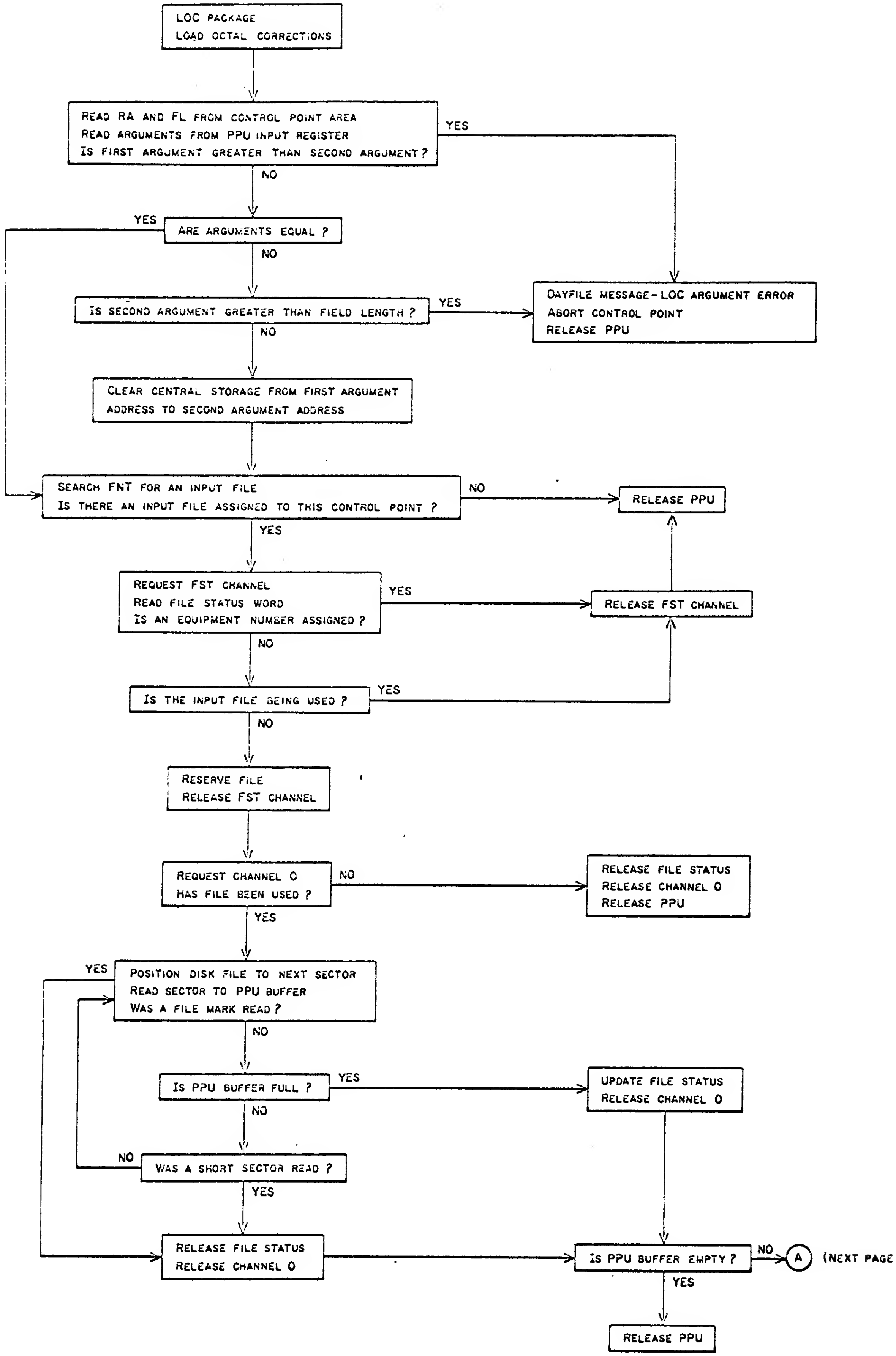
## LOC Routines

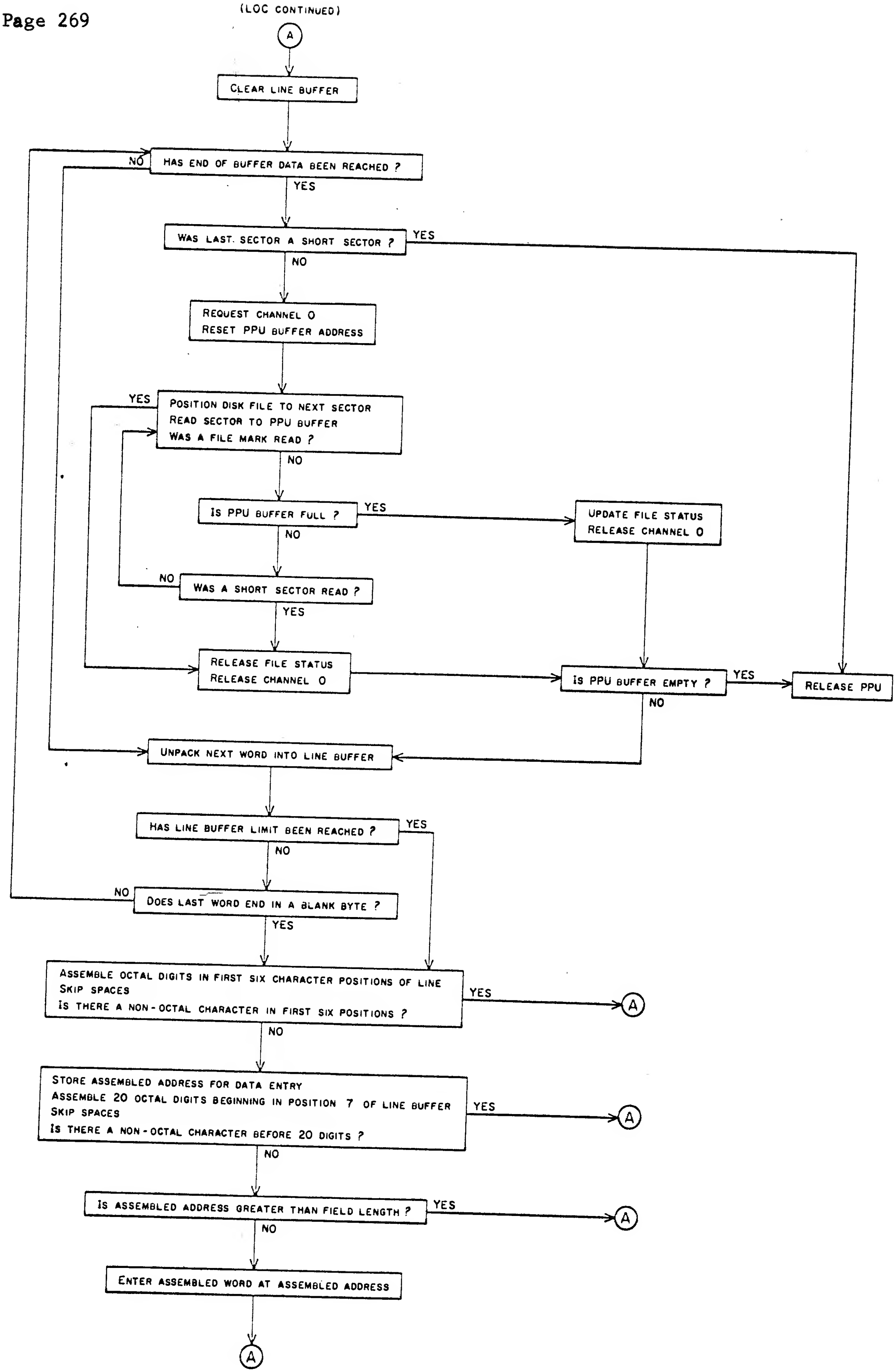
| | | |
|---|---|---|
| 1000 | Main Program | 1500, 1100, 1200, 1400, 1600 |
| | | 1300, 12-760, 100 |
| 1100 | Search For Input File | 12-760, 100, 740, 750 |
| 1200 | Load Buffer | 740, 700, 400, 750 |
| 1300 | Assemble Word | |
| 1400 | Unpack Character String | |
| 1500 | Clear Storage | 530, 13-760 |
| 1600 | Assembled Address | |

## Direct Core Cells

| | | |
|---|---|---|
| 1000 | P20/24 | FST entry |
| | P40/44 | Assembled word |
| | P50/54 | Input register |
| | P55 | RA (in hundreds) |
| | P56 | FL (in hundreds) |
| | P60 | Input buffer address |
| | P61 | Output buffer address |
| | P63/64 | Assembled address |
| | P75 | Input register address |
| 1100 | P01 | Local type and CP assignments |
| | P10/14 | FNT entry |
| | P20/24 | FNT status |
| | P20/24 | FST entry |
| | P50/54 | Input register |
| | P57 | FST address |
| 1200 | P01 | Sector length |
| | P06 | Track number |
| | P07 | Sector number |
| | P20/24 | FST entry |
| | P46 | Data byte |
| | P47 | Data byte |
| | P60 | Buffer input address |
| | P2000/7000 | Buffer |

| | | |
|---|---|---|
| 1300 | P01 | Octal digit |
| | P02 | Byte address |
| | P40/44 | Assembled word |
| | P62 | String address |
| 1400 | P60 | Input |
| | P61 | Output |
| | P62 | String address |
| | P7200/7400 | String buffer |
| 1500 | P10/14 | Zero word |
| | P50/54 | Input register |
| | P55 | RA |
| | P56 | FL |
| | P62/63 | First argument |
| 1600 | P01 | Octal digit |
| | 062 | String address |
| | P63/64 | Assembled address |


LOC PACKAGE
LOAD OCTAL CORRECTIONS
READ RA AND FL FROM CONTROL POINT AREA
READ ARGUMENTS FROM PPU INPUT REGISTER
IS FIRST ARGUMENT GREATER THAN SECOND ARGUMENT ?
YES
NO
ARE ARGUMENTS EQUAL ?
YES
NO
IS SECOND ARGUMENT GREATER THAN FIELD LENGTH ?
YES
NO
DAYFILE MESSAGE-LOC ARGUMENT ERROR
ABORT CONTROL POINT
RELEASE PPU
CLEAR CENTRAL STORAGE FROM FIRST ARGUMENT
ADDRESS TO SECOND ARGUMENT ADDRESS
SEARCH FNT FOR AN INPUT FILE
IS THERE AN INPUT FILE ASSIGNED TO THIS CONTROL POINT ?
NO
RELEASE PPU
YES
REQUEST FST CHANNEL
READ FILE STATUS WORD
IS AN EQUIPMENT NUMBER ASSIGNED ?
YES
RELEASE FST CHANNEL
NO
IS THE INPUT FILE BEING USED ?
YES
NO
RESERVE FILE
RELEASE FST CHANNEL
REQUEST CHANNEL 0
HAS FILE BEEN USED ?
NO
RELEASE FILE STATUS
RELEASE CHANNEL 0
RELEASE PPU
YES
YES
POSITION DISK FILE TO NEXT SECTOR
READ SECTOR TO PPU BUFFER
WAS A FILE MARK READ ?
NO
IS PPU BUFFER FULL ?
YES
UPDATE FILE STATUS
RELEASE CHANNEL 0
NO
NO
WAS A SHORT SECTOR READ ?
YES
RELEASE FILE STATUS
RELEASE CHANNEL 0
IS PPU BUFFER EMPTY ?
NO
A
(NEXT PAGE)
YES
RELEASE PPU


(LOC CONTINUED)
A
CLEAR LINE BUFFER
HAS END OF BUFFER DATA BEEN REACHED ?
NO
YES
WAS LAST SECTOR A SHORT SECTOR ?
YES
NO
REQUEST CHANNEL 0
RESET PPU BUFFER ADDRESS
POSITION DISK FILE TO NEXT SECTOR
READ SECTOR TO PPU BUFFER
WAS A FILE MARK READ ?
YES
NO
IS PPU BUFFER FULL ?
YES
NO
UPDATE FILE STATUS
RELEASE CHANNEL 0
WAS A SHORT SECTOR READ ?
NO
YES
RELEASE FILE STATUS
RELEASE CHANNEL 0
IS PPU BUFFER EMPTY ?
YES
NO
RELEASE PPU
UNPACK NEXT WORD INTO LINE BUFFER
HAS LINE BUFFER LIMIT BEEN REACHED ?
YES
NO
DOES LAST WORD END IN A BLANK BYTE ?
NO
YES
ASSEMBLE OCTAL DIGITS IN FIRST SIX CHARACTER POSITIONS OF LINE
SKIP SPACES
IS THERE A NON-OCTAL CHARACTER IN FIRST SIX POSITIONS ?
YES
A
NO
STORE ASSEMBLED ADDRESS FOR DATA ENTRY
ASSEMBLE 20 OCTAL DIGITS BEGINNING IN POSITION 7 OF LINE BUFFER
SKIP SPACES
IS THERE A NON-OCTAL CHARACTER BEFORE 20 DIGITS ?
YES
A
NO
IS ASSEMBLED ADDRESS GREATER THAN FIELD LENGTH ?
YES
A
NO
ENTER ASSEMBLED WORD AT ASSEMBLED ADDRESS
A

ROUTINE: MSG - Dayfile Message

PURPOSE: To enter messages from a central memory program into the dayfile.

GENERAL: This package checks the legality of the characters to be displayed and transmits them from central memory to this PP's message buffer. The lower 18 bits of the input register contains a beginning address of the message to be displayed.

METHOD:

1. The field length from CP(20) is read and the argument address of the message must be in bounds.

2. The message is checked character by character for legal display codes ($0\text{-}60_8$) and if all are legal, they are stored in the message buffer area of the PP.

3. A dayfile message "MESSAGE FORMAT ERROR" appears if:

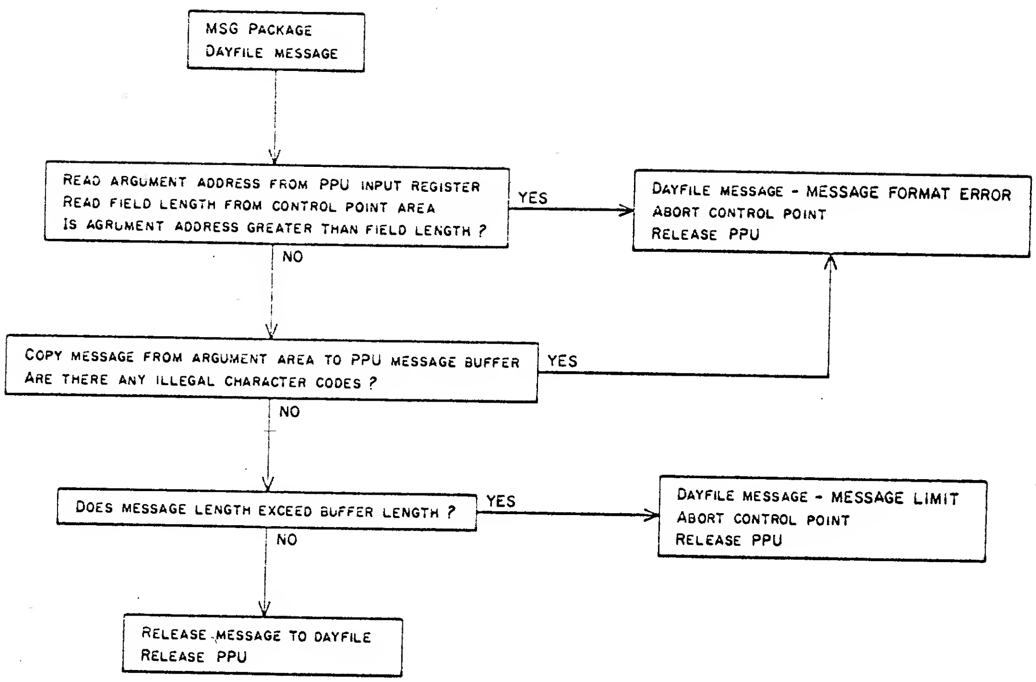
   a) The argument address is not within the field length.

   b) There is an illegal character in the message.

   c) The message length is greater than 6 central memory words.

4. In CP(22) there is a count of the total number of messages sent to the dayfile from the job assigned to this control point. If more than $100_8$ messages have been sent, a dayfile message of "MESSAGE LIMIT" appears and CP aborted.

5. A MTR code of 01 (dayfile message) is sent to the PP resident and after it has been processed, the PP is released.


MSG PACKAGE
DAYFILE MESSAGE
READ ARGUMENT ADDRESS FROM PPU INPUT REGISTER
READ FIELD LENGTH FROM CONTROL POINT AREA
IS AGRUMENT ADDRESS GREATER THAN FIELD LENGTH ?
YES
DAYFILE MESSAGE - MESSAGE FORMAT ERROR
ABORT CONTROL POINT
RELEASE PPU
NO
COPY MESSAGE FROM ARGUMENT AREA TO PPU MESSAGE BUFFER
ARE THERE ANY ILLEGAL CHARACTER CODES ?
YES
NO
DOES MESSAGE LENGTH EXCEED BUFFER LENGTH ?
YES
DAYFILE MESSAGE - MESSAGE LIMIT
ABORT CONTROL POINT
RELEASE PPU
NO
RELEASE MESSAGE TO DAYFILE
RELEASE PPU

ROUTINE: PBC - Punch Binary Cards.

PURPOSE: To format an area of central memory and punch it in the form of binary cards.

GENERAL: This package may be called by a control card or DIS console. Four calls may be issued:

1. no parameters - a binary deck beginning at RA and terminating one address less than the field length specified in the first word of the program. This call may be used to punch either a central or peripheral program in binary form.

2. one argument - area between RA and the address are punched.

3. two arguments - first argument is initial address and second is terminal address for a binary deck.

4. flagged - $400000_8$ argument - initial address specified by $400000_8$ + address. Lower 18 bits of this address added to it to form terminal address.

METHOD:

1. The initial address for the binary deck is read from the input register.

2. A check is made for the special $400000_8$ call. If the eighteenth bit of the terminal address is set, then the lower portion of the address (that left after $400000_8$ is subtracted) is set as the initial address. The lower 17 bits of this location is added to the initial address and used as the terminal address for the binary deck. Therefore, only a limited amount of memory may be punched if the 18th bit flag is set.

3. If the initial address is greater than the terminal address, the package is released without a diagnostic.

4. If the initial and terminal addresses are equal, then the lower 18 bits of RA is used as a terminal address. The initial address is cleared so that the area between RA and the FL-1 will be punched.

5. When the initial and terminal addresses have been set up properly, MTR is requested to assign the card punch to this job. If no card punch is available, the processing must wait on assignment.

6. Card punch assignment causes channel and synchronizer references within the package to be modified according to the entries from the EST.

7. Since the card punch is generally the slowest piece of equipment and PBC retains control of the PP until the complete binary deck is punched, a pause for MTR to adjust RA and FL during storage move is issued after every card is punched.

8. "PBC RANGE ERROR" and control point abort result if the terminal address ever becomes greater than the field length.

9. The punch buffer is loaded with data for the next card. In column one is stored 7-9 punches and card length. The data bytes are summed and stored in column two module 4095. Column 79 is not used and the binary sequence number is stored in column 80.

10. The punch must be ready or a console message "PUNCH NOT READY" is sent.

11. One card is then punched.

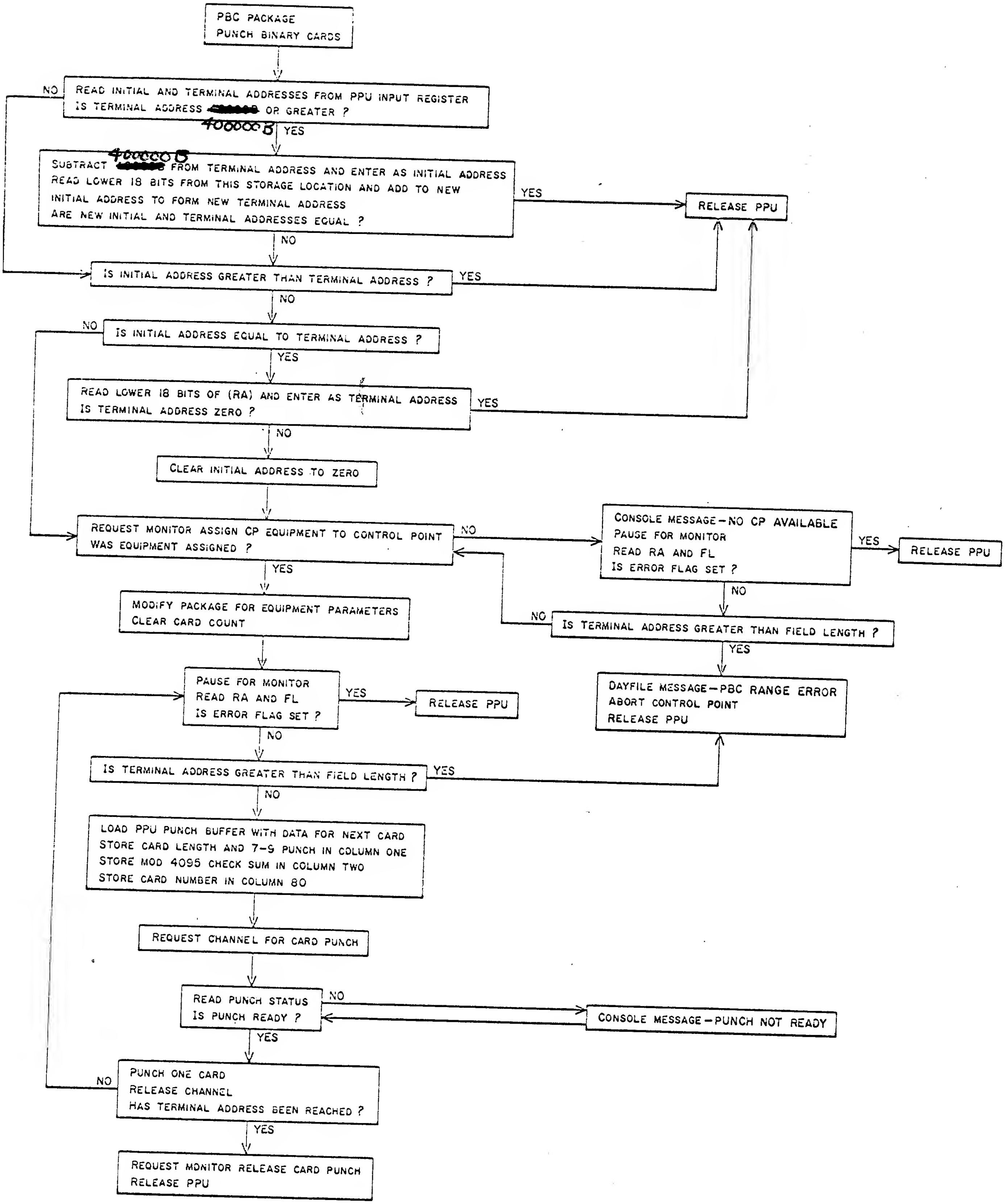
12. When the terminal address is reached, the package is released so that one less than the terminal address words are punched.

NOTES:

1. The flagged call is used by the Fortran compiler to punch a deck in I mode.

PBC Routines

| | | |
|---|---|---|
| 1000 | Main Program | 1640, 1600, 1200, 1500, 1100, 1300, 1400, 23-760, 12-760, 100 |
| 1100 | Sense CP status | 17-760, 530, 13-760, 12-760, 100 |
| 1200 | Request CP | 22-760, 1100 |
| 1240 | Sense punch ready | |
| 1300 | Load Punch buffer | |
| 1400 | Punch one card | 13-740, 1240, 1440, 750 |
| 1440 | Output one byte to punch | |
| 1500 | Modify program for equipment parameters | |
| 1540 | Channel modification table | |
| 1560 | Synchronizer modification table | |
| 1600 | Process RA length | 12-760, 100 |
| 1640 | Process flag length | |



PBC PACKAGE
PUNCH BINARY CARDS
READ INITIAL AND TERMINAL ADDRESSES FROM PPU INPUT REGISTER
IS TERMINAL ADDRESS [redacted] OR GREATER ?
400000B
NO
YES
SUBTRACT 400000B [redacted] FROM TERMINAL ADDRESS AND ENTER AS INITIAL ADDRESS
READ LOWER 18 BITS FROM THIS STORAGE LOCATION AND ADD TO NEW
INITIAL ADDRESS TO FORM NEW TERMINAL ADDRESS
ARE NEW INITIAL AND TERMINAL ADDRESSES EQUAL ?
YES
RELEASE PPU
NO
IS INITIAL ADDRESS GREATER THAN TERMINAL ADDRESS ?
YES
NO
IS INITIAL ADDRESS EQUAL TO TERMINAL ADDRESS ?
NO
YES
READ LOWER 18 BITS OF (RA) AND ENTER AS TERMINAL ADDRESS
IS TERMINAL ADDRESS ZERO ?
YES
NO
CLEAR INITIAL ADDRESS TO ZERO
REQUEST MONITOR ASSIGN CP EQUIPMENT TO CONTROL POINT
WAS EQUIPMENT ASSIGNED ?
NO
CONSOLE MESSAGE—NO CP AVAILABLE
PAUSE FOR MONITOR
READ RA AND FL
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
NO
IS TERMINAL ADDRESS GREATER THAN FIELD LENGTH ?
YES
DAYFILE MESSAGE—PBC RANGE ERROR
ABORT CONTROL POINT
RELEASE PPU
YES
MODIFY PACKAGE FOR EQUIPMENT PARAMETERS
CLEAR CARD COUNT
PAUSE FOR MONITOR
READ RA AND FL
IS ERROR FLAG SET ?
YES
RELEASE PPU
NO
IS TERMINAL ADDRESS GREATER THAN FIELD LENGTH ?
YES
NO
LOAD PPU PUNCH BUFFER WITH DATA FOR NEXT CARD
STORE CARD LENGTH AND 7-9 PUNCH IN COLUMN ONE
STORE MOD 4095 CHECK SUM IN COLUMN TWO
STORE CARD NUMBER IN COLUMN 80
REQUEST CHANNEL FOR CARD PUNCH
READ PUNCH STATUS
IS PUNCH READY ?
NO
CONSOLE MESSAGE—PUNCH NOT READY
YES
PUNCH ONE CARD
RELEASE CHANNEL
HAS TERMINAL ADDRESS BEEN REACHED ?
NO
YES
REQUEST MONITOR RELEASE CARD PUNCH
RELEASE PPU

# Program Partitioning

## I. Introduction

Chaining is a method used to execute a program which exceeds available storage or field length. The program is separated into a main program and any number of segments which may be called and executed as needed by the FORTRAN program. Both the main program and segments may contain one or more subroutines and/or functions. Overlays may be loaded (and executed) or replace the calling program by appropriate central program machine language action.

## II. RUN Modes

A copy of the compiled program or segment(s) is always left on the disk. Either may be called (by name) and executed separately. Each partition (segment) including its subroutines must be separated from the main program or other partitions by a record separator. Two consecutive record separators must separate the last END statement from the first data card or file separator.

### A. Chain Mode -- RUN(C,........)

Chain mode is comparable to 6 mode except that segments may be assembled following the main program. That is, no listing is produced and execution is assumed unless compile errors are encountered. The programs to be compiled must be a PROGRAM followed by one or more SEGMENT(s) each separated by a record separator.

### B. Batch Mode -- RUN(B,........)

Batch mode is comparable to S mode except that any combination of one or more programs, subroutines, segments, or functions may be compiled. Also, a listing of the source language is always produced and execution is not assumed. Each program and segment is written on the disk as a file using the name specified on the PROGRAM or SEGMENT card. Therefore, execution may be initiated by a Program Call Card.

## III. FORTRAN Usage

### A. Definition of Segment

Each segment must begin with the statement:

SEGMENT name $(f_1, f_2, f_3, \ldots, f_n)$

where name is an alphanumeric identifier for the segment. This is the name that must be used when calling the segment

$f_1, f_2$, ..are file names of the files used any place in the program. These file names must agree in number and order with those specified for the main program. All files used in the execution of the main program and all segments must be specified on the PROGRAM and all SEGMENT cards.

Compilation of segments and programs differ only in the following respect:

1. Blank common is not cleared to zero by the object code in a segment.

2. Buffer space and parameters are not initialized by the object code in a segment. They are carried over from the main program in order not to destroy any input or output when calling segments.

### B. Calling a Segment

A segment is called by using the FORTRAN statement:

```
CALL    CHAIN    (name)
```

Where CHAIN is the subroutine that loads and initializes execution of the called segment.

name is the identifier of the segment to be loaded and executed.

Segments to be called by CHAIN may reside as a named file on the disk. The only parameter to CHAIN must be the segment name.

### C. General

1. Segments may be called from either the main program or another segment.
2. Calling of a segment causes the segment to be loaded over the calling program thus destroying the main program or segment that issues the call.
3. Segments may be called more than once.
4. Parameters and communication between segments can be passed only through the use of blank common.

5. Each segment is compiled beginning with relative address zero (RA = 0).
6. In order to match locations of blank common, all elements of blank common must be described in the same order and number in the main program and all segments <u>or</u> the length of common must be declared on the RUN card.

Example:

```
        CHNTST, 1, 100, 40000
        MODE 7.
        RUN (B)
        CHN.
        7-8-9
*                          PROGRAM CHN (INPUT, OUTPUT, TAPE10)
**                         COMMON  I, J, K, A(5), B(10)
                           READ 5, A
                                   .
                                   .
                                   .
                           CALL CHAIN (S2)
                           END
        7-8-9
*                          SEGMENT S1 (INPUT, OUTPUT, TAPE 10)
**                         COMMON  I, J, K, A(5), B(10)
                                   .
                                   .
                                   .
                           WRITE (999, 10) B(10)
                                   .
                                   .
                                   .
                           CALL CHAIN (S3)
                           END
        7-8-9
*                          SEGMENT S2 (INPUT, OUTPUT, TAPE10)
**                         COMMON  I, J, K, A(5), B(10)
                                   .
                                   .
                                   .
                           CALL CHAIN (S1)
                           END
        7-8-9
*                          SEGMENT S3 (INPUT, OUTPUT, TAPE10)
**                         COMMON  I, J, K, A(5), B(10)
                                   .
                                   .
                                   .
                           END
        7-8-9
        7-8-9

                           Data Deck
      6-7-8-9
```

* These statements must specify all file names even though they are not referenced in the segment or program.

* All elements must be included in the list.

## IV. Machine Language Calls

Two peripheral packages are available for loading and/or executing segments. One loads one or more segments. The other loads and executes one segment or program destroying the calling program.

### A. EXU

This package loads a program to replace the calling program and initiates execution of the loaded program. The calling program is destroyed.

#### 1. CALL

The routine is called by setting certain parameters into RA+1 of the calling program.

```
RA+1 = EXU00.........0LLLLLL

       18     24       18       bits
```

when EXU is in display code,

LLLLL is the address of the argument. The argument is the name of the central program to be loaded and executed. The name is specified in display code with trailing spaces.

#### 2. Usage

After the monitor recognizes the request in RA+1 and assigns a PPU to process the request, RA+1 is cleared to zero by the PPU. At this point, the central program must terminate itself normally in order to allow the PPU to load the program. The central program is terminated by placing END (trailing spaces) in RA+1 and looping until it is terminated.

EXU resets or clears all operational registers - $A_n$, $B_n$, $X_n$ - before executing the called program.

EXU loads only from job files on disk 0 (common or local)

#### 3. Example:

Following is an ASCENT subroutine which may be called from a FORTRAN program to call EXU. This example is very similar to the CHAIN subroutine except the name of the program is fixed to SEG1.

```
Col.  2      7       11

             ASCENTF  SUBROUTINE  LDS
                      PS
                      PS
      EXIT            PS
      TAG1            SA1 = 1
```

```
                NZ  X. TAG1     .ASSURE RA+1 = 0
                SX6=053025B
                LX6 42
                SX1=SEG1
                IX6=X6+X1
                SA6=1           .SET RA+1 TO EXU PARAMETER
TAG2            SA2=1
                NZ  X2 TAG2     .WAIT FOR PPU TO ACCEPT CALL
                SX7=051604B
                LX7 42
                SA7=1           .SET RA+1 TO END
TAG4            ZR  BO BO TAG4  .WAIT FOR THE PROGRAM TO TERMINATE
          ..
SEG1            CON 23050734000000000000B
          END
```

B. CLL

This package loads one or more central programs or segments into an area of memory specified by the calling program.

1. Call

This routine is called by setting certain parameters into RA+1 of the calling program.

```
RA+1 = CLL 0.....0 BA
       18     24   18    bits
```

| | | |
|---|---|---|
| BA | | FIRST |
| BA+1 | | LIMIT |
| BA+2 | PROG 1 | P1 |
| BA+3 | PROG 2 | P2 |
| | | |
| BA+n+1 | PROG n | Pn |
| BA+n+2 | (zero) | |

where CLL is in display code
BA is an 18 bit address where the parameters are located
FIRST is the beginning address for loading the first program.
LIMIT is the limit address for loading the programs
PROG1
PROG2
PROGn are the names (in display code with trailing spaces) of the programs or segments to be loaded.
P1,P2
..,Pn are set by CLL after loading the programs and are the beginning addresses of the associated overlays.

All of the parameters except Pn must be set up by the calling program prior to setting RA+1.

2. Usage

CLL loads the programs one at a time beginning with the name specified at BA+2. The order of search for locating the overlays is:

1. Resident Subroutine Library - RSL
2. Central Library Directory - CLD
3. Assigned Job Files - common or local

The programs are loaded into the consecutive memory locations beginning with FIRST. No program may be loaded beyond the address specified by LIMIT. After a program is loaded, its beginning address is entered into the lowest 18 bits of the respective parameter word. After Cll has completed the call, BA is cleared to zero.

If program cannot be located, the address Pn for the program is not modified by CLL. If a program exceeds LIMIT, the value 777777 is entered into the respective address Pn. The last parameter must be followed by a full word containing zero.

It should be remembered that programs and segments compile with a reference address beginning with zero (000000). Since the central program calling CLL resides at zero, the loaded programs (by CLL) will not have proper address terms for those instructions containing 18 bit address. Therefore, the user must modify the addresses of the loaded program or use some addressing scheme where the calling program defines a pseudo-reference address in an index register whenever memory is referenced.

CONTROL DATA CORPORATION

Development Division - Applications

# CIRCULAR INPUT OUTPUT

Chippewa Operating System

10/20/65

# CIRCULAR INPUT OUTPUT

# CIO

## INTRODUCTION

All input and output for a file is passed through a circular central memory buffer. Buffer parameters are initialized by the central memory program and then the CIO package is called to perform the transfer to or from the physical medium of the file. These parameters are altered by CIO or the central program as data is inserted or extracted from the buffer. A circular effect is achieved by allowing the data to wraparound the buffer whenever the limit address of the buffer is reached. For example, on an input request data is inserted into contiguous words until the last address of the buffer is encountered. The next piece of data will be stored in the beginning address of the buffer so that the total capacity of the buffer may be utilized. All system central memory buffers, i.e. dayfile, etc., use this circular motion even if CIO is not specifically called to perform the I/O operation.

## CALLING SEQUENCE

A program requesting I/O must set up certain buffer parameters. The location of these parameters is sent to CIO via the lower 18 bits of RA+1. These parameters, along with the buffer itself must reside within the field length of the job, and their addresses are relative to RA.

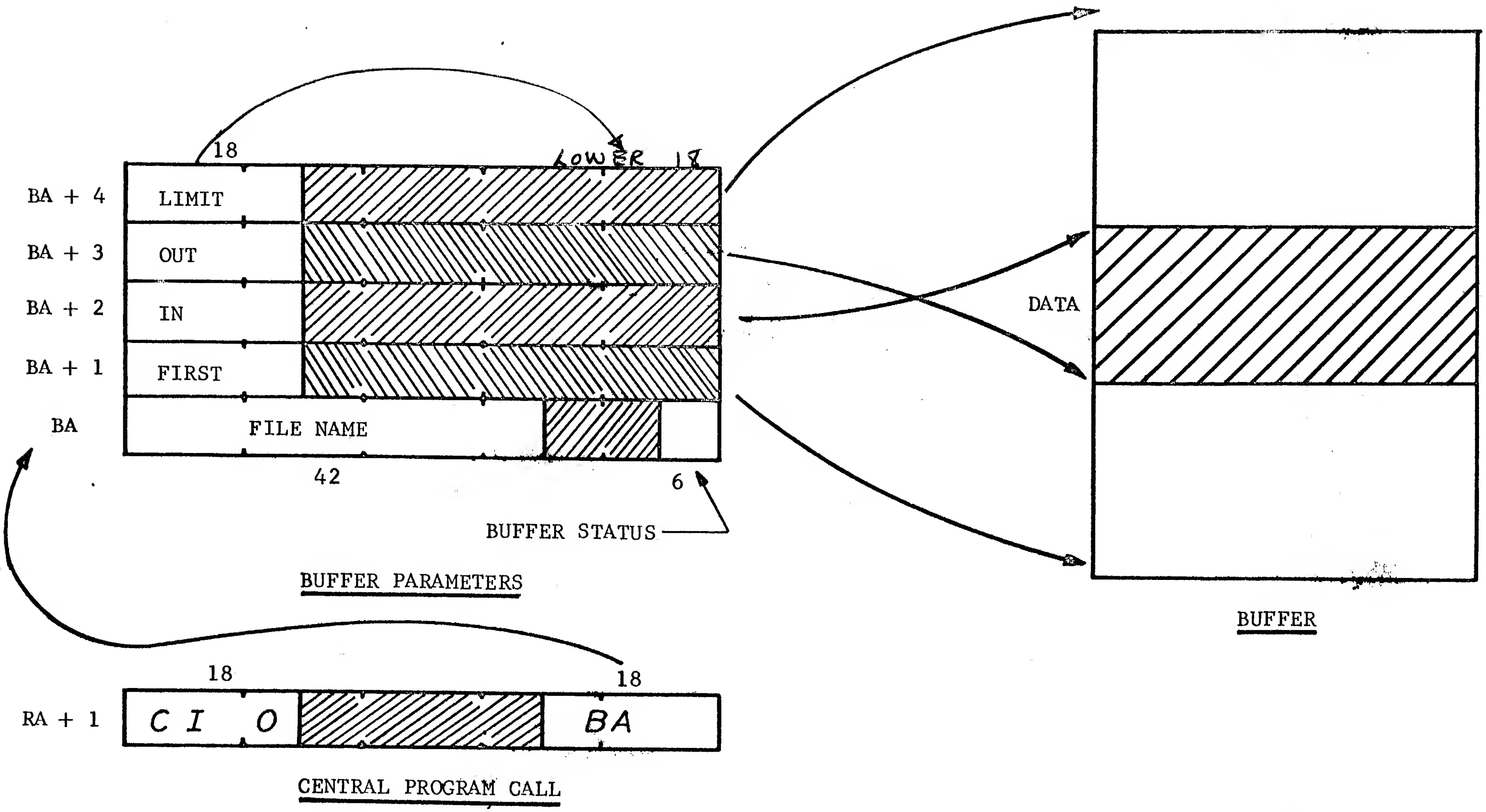
Five central memory words, designated as BA to BA+4, hold the parameters. In the first word is the name of the file in left-justified display code to be acted upon and a six bit code called the buffer status. The first digit of the buffer status specifies the type of operation: the second gives the direction (read/write) and the mode (coded/binary).

BA+1 contains the beginning address of the buffer and is called FIRST. Along with LIMIT, the last address of the buffer plus one, FIRST remains dormant, i.e. CIO never changes these values. No data is stored in LIMIT. When LIMIT is reached, the next available address for storage is FIRST. The buffer capacity is referred to as the area between FIRST and LIMIT-1.

The two remaining words, BA+2 and BA+3, are the actual pointer addresses. IN (BA+2) defines the next available address for insertion of data into the buffer. OUT (BA+3) holds the address for removal of data from the buffer. Therefore, the amount of data residing in the buffer is that between IN and OUT. IN is advanced around the buffer, but never passing OUT so as not to overstep the buffer capacity, by a 'read' operation. Any 'write' request causes OUT to move in the direction of IN and to pass data from the buffer to the file in its advance.

Either CIO or the central program may update IN and OUT. By moving IN, CIO could read data from a file to the buffer and the central program could remove the data from the buffer for its own use by moving OUT. The opposite effect would result if the central program inserted data into the buffer by incrementing IN and CIO transferred the buffer data to the file by moving OUT.

Initially, the buffer parameters are set FIRST = IN =OUT with IN and OUT circling the buffer as data is inserted or removed. An empty buffer is reflected by IN = OUT. This condition is distinguished from a full buffer, IN = OUT-1, by an unused word between IN and OUT. The useable data in the buffer begins at OUT and continues (circling the buffer if necessary) to IN-1.



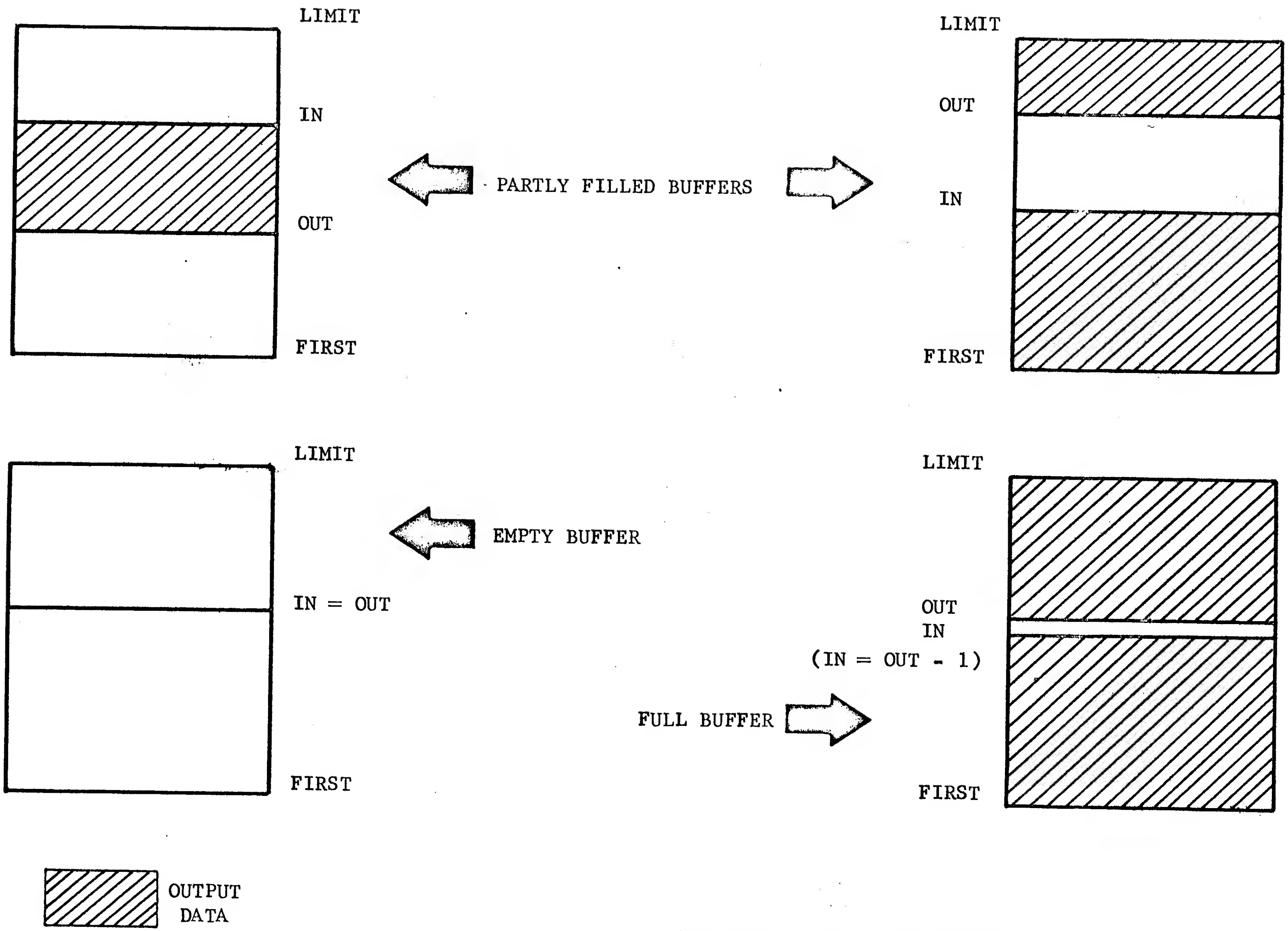

CIO PARAMETERS

BUFFER STATUS

The buffer status appears as a 2 digit octal code in the lower 6 bits of BA. This code indicates the mode of the buffer and provides an interlock for peripheral package activity. The buffer status has an even value when CIO is called. It is set to an odd value when the peripheral package has completed the I/O function. This six bit code is also kept as the last buffer status in the FST entry for the file. Whenever this value is checked and found to be even, the file is assumed to have an operation being performed on it, i.e. it is active. An odd value means that the file is not busy and is available for use.

Normal reads and writes use 'buffer I/O' as the type of operation (first octal digit with the second digit specifying the mode). This is interpreted by CIO as a request to transfer as many records as possible between the file and the buffer. A short record, i.e. end-of-file or end-of-record, or a full buffer will terminate a read operation and a write request is stopped whenever OUT = IN. If a read was requested, CIO will alter the code to indicate whether an 'end-of-record' or 'end-of-file' was read. Whenever the buffer is to be emptied to the file by a write operation, the central program must issue either an 'end-of-record' or 'end-of-file' write. This causes all of the data to be transferred and an 'end-of-logical-record' or 'end-of-file' to be written on the file. Therefore, if the buffer does not contain a full record of data and an 'end record' write was not issued, no data will be transferred.

When MTR accepts the I/O request by assigning CIO to a PP, RA+1 is cleared. In order for the central program to know when the PP has finished the I/O operation, the buffer status must be checked for an odd value.




CIRCULAR BUFFERS

If a 'read' was requested, the first octal digit may have been altered by CIO, but otherwise, only the second digit is incremented by one. On a 'binary backspace' the first digit is a 4 and the second may be either a 2 or 6 because only the second bit in the code is checked for the set condition.

## INTERNAL STRUCTURE

Whenever an I/O operation is to be performed on a file, the CIO package is assigned to a peripheral processor. CIO examines the request and passes control within itself to the proper function. The buffer parameters are checked for legality to insure that the operation remains within the job's field definition.

A file may be of any equipment type - disk, card reader, card punch, line printer, or magnetic tape. The driver for each operation on a piece of equipment is written as a separate routine. The CIO function decides which driver is needed and calls it into the PP as an overlay. These overlays do the physical I/O and update the buffer parameters accordingly. Whenever their task is finished, CIO completes the request so that the calling program may continue its execution.

## OPERATION

Each function ( read, write, or backspace) within CIO calls special overlays. These overlays do more specific parameter checking to insure that the buffer can contain the amount of data requested. Parity error checking and buffer status updating are also the responsibilities of the overlay.

The Read function calls 2RD (read disk), 2RT (binary tape read), 2RC (read cards), and 2RT (BCD tape read). IN is incremented to reflect the number of words read from the file to the buffer and the first octal digit of the buffer status may be changed if an 'end-record' or 'end-file' is read.

Data is read by 2RD one sector ($100_8$ central memory words) at a time until a short sector is encountered or the buffer is filled. If a disk parity error is found, the sector is reread with varying margins three times and if it persists the PP stops. Only dead start will force reinitialization.

2RC transfers only useable data to the buffer by suppressing trailing blanks with a zero byte ( 12 bits). Ten characters per word are translated from Hollerith to display code and packed until a zero byte is inserted to signal 'end-of-physical-record'. A 7-8-9 card causes a short sector to be transferred. Only one file mark may appear within one diskfile, so when a 6-7-8-9 card is found, an 'end-of-record' sector is copied along with a second short sector to indicate end-of-file.

A binary record of less then 4 bytes is considered a noise record by 2RT. If this overlay discovers that there is not enough room in the buffer to handle a full block of 512 words, no data is transferred. A read is tried 3 times before a parity error message is sent to the dayfile. Only one block of data is read per request. Rewinding is also done by this overlay.

A BCD tape record is a constant 120 characters long. All trailing spaces are eliminated by a zero byte and the BCD characters are translated into display code. A record less than 6 bytes is considered noise and a record is read 3 times before a parity error message is sent.

The Write function is in-charge-of updating OUT. As data is removed from the buffer and copied to the file, OUT moves in the direction of IN until OUT = IN. Only a short record request will cause the buffer to be completely dumped of information and the appropriate indicator to be written on the file.

A check is made to see if the last reference to a disk file was a write. If it was not, the tracks thus far reserved by the file are dropped by 2DT. This provides multi-use of a file. Data written on a file can be backspaced and read and another write request will cause the beginning of the file to be referenced.

2WD is loaded to write disk data. If there is not enough data in the buffer for a full sector (64 words) and an 'end-record' was not requested, no data is written on the file. Every write is terminated by an EOF sector but since none of the parameters are advanced, the next write request will write over this sector. It prevents a file from ever running away. Two tracks are requested at once so that time is not wasted whenever one track is filled and another is needed.

To punch both binary and Hollerith cards, 2PC is loaded. This overlay is called whenever a file has been assigned to the card punch by a control card and a write operation requested on the file. Eighty characters or the number of characters to the first zero byte are assembled from display code to Hollerith. A 7-8-9 or 6-7-8-9 card is punched if requested. In the case of a binary request, 15 words of data are punched on a card with the appropriate binary controls - word count, 7-9 punches in column one, checksum, and sequence number.

2LP is loaded to print the file assigned to a line printer. A print line consists of either 130 characters or the number of characters to a zero byte. Page spacing is checked by this overlay.

To write a binary tape 2WT is loaded. This overlay is called into play to do all writes on 1" tape and binary writes on ½" tape. Coded records on 1" tape are in packed display code and terminated by a zero byte. A logical record consists of $1000_8$ central memory words. If a parity error is encountered, the tape is backspaced and rewritten with no erasing until a good write is made or an error flag is set. 2WT also writes a file mark when one is requested.

2WT is loaded to write BCD tape. All BCD tape records are 120 characters long. If a zero byte is found before 120 characters have been converted from display code to BCD, the record is padded with spaces until 120 characters are reached. The writing continues for full blocks of data contained in the buffer until an 'end-record' or 'end-file' is requested to empty the buffer.

The Backspace function is called to backspace either binary or coded records. An end-of-file is considered a record or a coded line in each mode respectively. This action causes IN to be advanced down the buffer and a read is not necessary after a backspace to make the data available for use. A binary disk backspace may be very slow. Since a record can be written on several tracks, each pointer word before each sector must be checked for a track change. If a file contains only one record, a rewind operation is much faster.

2BD does either binary or coded backspacing on the disk. A binary backspace is done until a short sector is found. The file will be

positioned either in front of the file mark just written or at the beginning of the last record. Only one coded line is backspaced with this request. OUT will reflect the address before the last card image or zero byte. No read is required to bring the data back in because the pointer words are properly adjusted.

2BT is loaded to perform the same backspace operations on tape. The physical tape is moved.

RECALL

The central program retains control of the central processor while CIO is performin the I/O operation. MTR clears RA+1 when the CIO request is accepted informing the central program to continue processing. If no further processing can be done until the data is transferred, the central processor should be given to another job. By inserting an RCL call (recall) in RA+1, control is taken away from the central program by MTR and switched to another job. Control is regained when a PP completing an operation tells monitor to recall the proper central program, or a time span of near 250 ms. has lapsed. Effective use of recall allows the central processor to be utilized more efficiently.

A workable sequence of events that will allow the central processor to execute other jobs while an I/O operation is holding up a central program is:

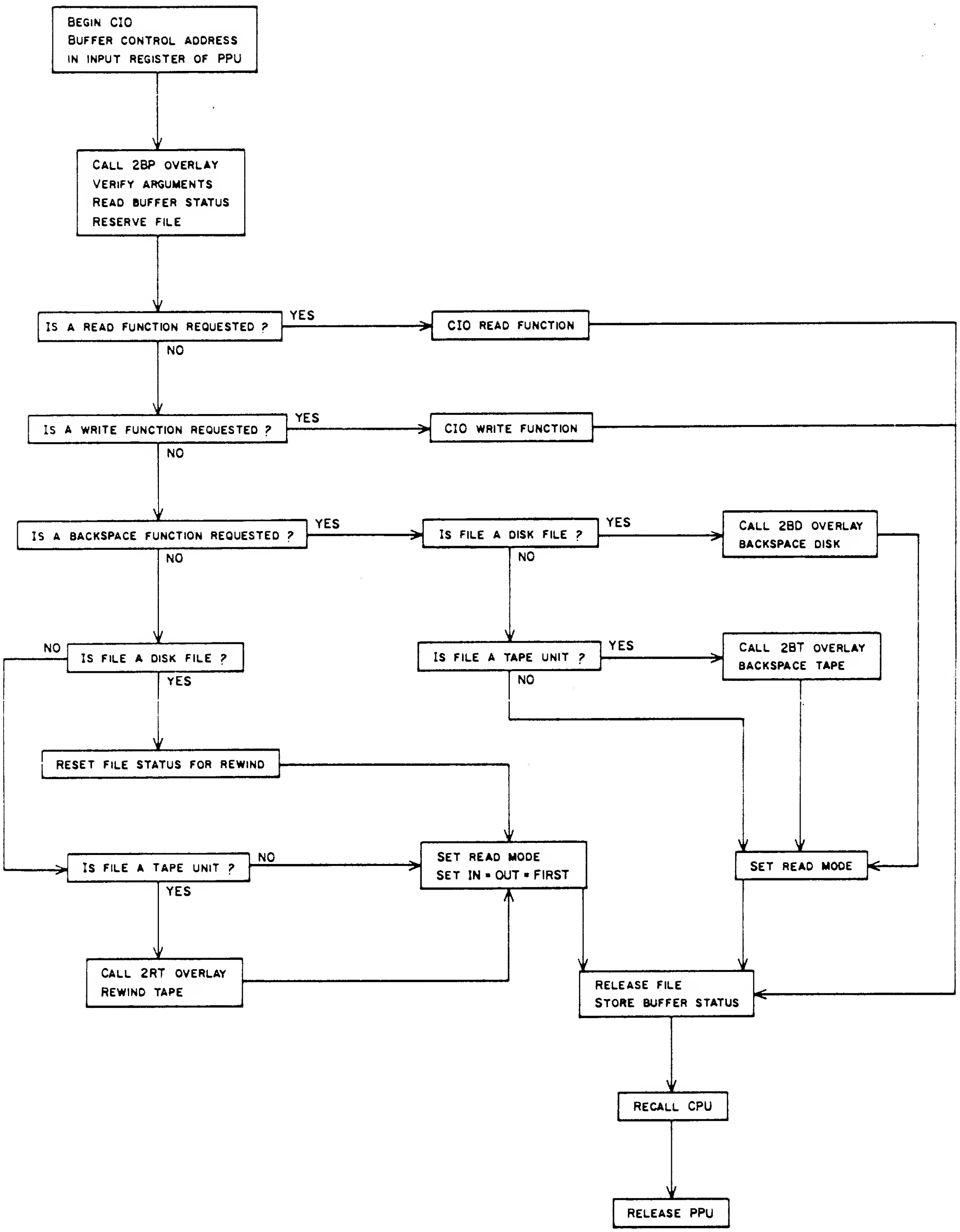
1) Send CIO call to RA+1
2) Wait until MTR has accepted the request by clearing RA+1
3) Check buffer status for an odd value.
4) If an odd value is found, continue normal processing, otherwise send RCL call to RA+1
5) Repeat steps 2-4, exiting only if the buffer status is odd

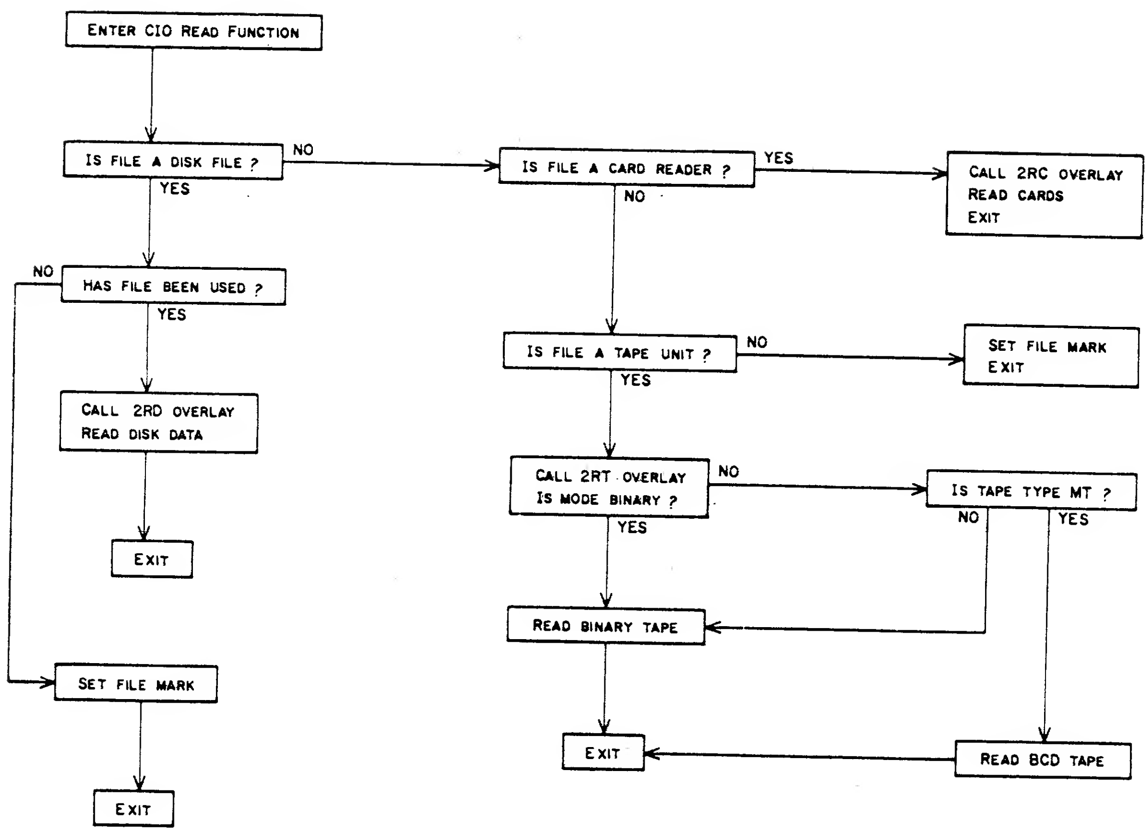
## BUFFER STATUS

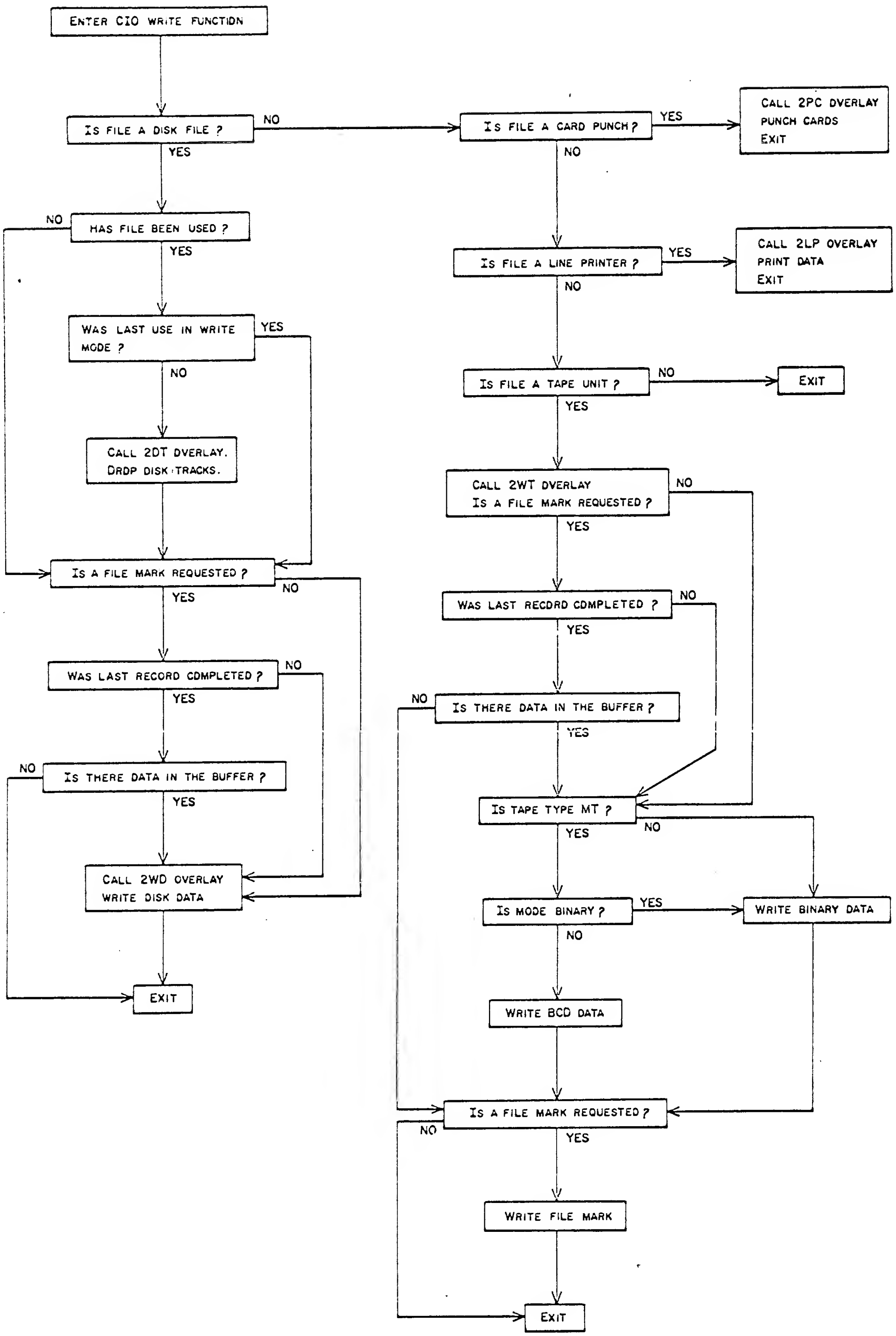
| first digit | | second digit | |
|---|---|---|---|
| 0X | not used | X0 | request coded read |
| 1X | buffer I/O | X1 | completed coded read |
| 2X | end record | X2 | request binary read |
| 3X | file mark | X3 | completed binary read |
| 4X | backspace | X4 | request coded write |
| 5X | rewind | X5 | completed coded write |
| 6X | rewind/unload | X6 | request binary write |
| 7X | not used | X7 | completed binary write |

EXAMPLE:

| | | | |
|---|---|---|---|
| READ | request to CIO | 10 | |
| | answer from CIO | 11 | full buffer |
| | | 21 | end of record encountered |
| | | 31 | end-of-file encountered |
| WRITE | request to CIO | 14 | dump as many complete records as possible |
| | | 24 | empty buffer and write end of record |
| | | 34 | empty buffer and write end of file |
| | answer from CIO | X5 | where X from the call is unaltered |


BEGIN CIO
BUFFER CONTROL ADDRESS
IN INPUT REGISTER OF PPU
CALL 2BP OVERLAY
VERIFY ARGUMENTS
READ BUFFER STATUS
RESERVE FILE
IS A READ FUNCTION REQUESTED ?
YES
NO
CIO READ FUNCTION
IS A WRITE FUNCTION REQUESTED ?
YES
NO
CIO WRITE FUNCTION
IS A BACKSPACE FUNCTION REQUESTED ?
YES
NO
IS FILE A DISK FILE ?
YES
NO
CALL 2BD OVERLAY
BACKSPACE DISK
IS FILE A DISK FILE ?
NO
YES
IS FILE A TAPE UNIT ?
YES
NO
CALL 2BT OVERLAY
BACKSPACE TAPE
RESET FILE STATUS FOR REWIND
IS FILE A TAPE UNIT ?
NO
YES
SET READ MODE
SET IN = OUT = FIRST
SET READ MODE
CALL 2RT OVERLAY
REWIND TAPE
RELEASE FILE
STORE BUFFER STATUS
RECALL CPU
RELEASE PPU


ENTER CIO READ FUNCTION
IS FILE A DISK FILE ?
NO
YES
HAS FILE BEEN USED ?
NO
YES
CALL 2RD OVERLAY
READ DISK DATA
EXIT
SET FILE MARK
EXIT
IS FILE A CARD READER ?
YES
NO
CALL 2RC OVERLAY
READ CARDS
EXIT
IS FILE A TAPE UNIT ?
NO
YES
SET FILE MARK
EXIT
CALL 2RT OVERLAY
IS MODE BINARY ?
NO
YES
IS TAPE TYPE MT ?
NO
YES
READ BINARY TAPE
EXIT
READ BCD TAPE


ENTER CIO WRITE FUNCTION
IS FILE A DISK FILE ?
NO
YES
HAS FILE BEEN USED ?
WAS LAST USE IN WRITE MODE ?
CALL 2DT OVERLAY. DROP DISK TRACKS.
IS A FILE MARK REQUESTED ?
WAS LAST RECORD COMPLETED ?
IS THERE DATA IN THE BUFFER ?
CALL 2WD OVERLAY WRITE DISK DATA
EXIT
IS FILE A CARD PUNCH ?
CALL 2PC OVERLAY PUNCH CARDS EXIT
IS FILE A LINE PRINTER ?
CALL 2LP OVERLAY PRINT DATA EXIT
IS FILE A TAPE UNIT ?
EXIT
CALL 2WT OVERLAY IS A FILE MARK REQUESTED ?
WAS LAST RECORD COMPLETED ?
IS THERE DATA IN THE BUFFER ?
IS TAPE TYPE MT ?
IS MODE BINARY ?
WRITE BINARY DATA
WRITE BCD DATA
IS A FILE MARK REQUESTED ?
WRITE FILE MARK
EXIT

CONTROL DATA CORPORATION

Development Division - Applications

# DAYFILE

Chippewa Operating System

10/20/65

# DAYFILE

## INTRODUCTION

The dayfile is a combination accounting medium and job status record. It appears as a major display for the system console and is part of every job's output. Any message a programmer wishes to convey to an operator is passed through the dayfile. All control cards, error diagnostice, running times, and equipment assignments appear as a console display and are later sorted for a particular job's output.

A message may enter the dayfile from a central memory program or a peripheral routine. In the case of a central memory program, a peripheral package (MSG) is called to transfer the message from central memory to the PP message buffer and then to inform monitor that dayfile action is required. A peripheral routine need only put the message in the message buffer and let monitor take the appropriate steps. When monitor does sense that a message is ready, it transfers the message to the associated control point's dayfile area and then sends the message to the dayfile buffer in the proper format. This new entry is then picked up by the display program (DSD) and shown on the console.

## STRUCTURE

The dayfile buffer status (DFB) is contained in word three of central memory. It points to a $1000_8$ word buffer for dayfile entries and maintains the FIRST, IN, OUT, and LIMIT addresses. Each entry made in the dayfile consists of three parts.

1) The time that a message is sent.
2) The name of the job to which the message belongs.
3) The message of not more than six words.

All three parts are in separate words. Therefore, every dayfile entry is at least three words long but not more than nine. The time is read from word thirty of central resident and is in the form XX.YY.ZZ. where XX is hours, YY is minutes, and ZZ is seconds. At dead start this word is zeroed so it will reflect the time since dead start unless a "TIME" entry is made to DSD via the keyboard.

Monitor changes the spaces in the job name to blanks and terminates the field with a period. A zero byte ends the message so that word after word is transferred until the zero byte is found.

UPDATING BY CENTRAL PROGRAMS

In order for a central program to make entries into the dayfile, a peripheral program (MSG) is called to retrieve the message from central memory and inform monitor of the request. The location of the message and MSG (in left-justified display code) is inserted in RA+1 of the program. This causes MSG to be assigned to a PP and the message transferred to the PP's message buffer.

A check is made to insure that every character is a legal display code. If an illegal character is found, MESSAGE FORMAT ERROR is issued to the dayfile and the job is abandoned. Every entry made into the dayfile by a particular job advances a message count by one. In MSG this total is examined for an excess of $100_8$ messages. No more than 6 words may be passed to the dayfile in one message. If either of these rules is violated, MESSAGE LIMIT is sent to the dayfile.

After the message is residing in the PP's message buffer, MTR is informed so that the message can be passed to the dayfile buffer. MSG is used by the Fortran compiler to enter the name of the program currently being compiled or executed.

UPDATING BY PERIPHERAL PROGRAMS

In a peripheral processor's resident program is a section of coding which copies a message from a transient program into the PP's message buffer. Each of these messages is assumed to have legal display codes and ended by a zero byte. The location of the message is in the A register upon entry to the routine. A return jump to location 530 will cause the message to be transferred to the PP's message buffer and then MTR is told of the request.

All transient programs use this method of making entries into the dayfile and each request will advance the message count at a job's control point, even though MSG is the only program which checks for a excess of the limit.

MTR - ISSUE DAYFILE

Whenever a PP has a message for the dayfile, the message is put into the message

buffer and 0001 is inserted into the first byte of output register. MTR senses a request and begins dayfile updating procedures.

The message is passed from the PP buffer to an eight word area in the control point area. Word 30 of central memory which contains the current time is read into one word and the name of the job is put into another word. Next the message is copied until a zero byte is encountered and then all three sections are sent to the dayfile buffer. The PP output register is cleared to inform the PP that the message has been transferred. Only at this pointe is the message count increased by one so that every message is totalled.

IN and OUT are checked to see if $100_8$ words (a full sector) of information is contained in the day file buffer. If there is, phase one dump flag is set. No additions may be made to the buffer when a dump flag is set.

## MTR - COMPLETE DAYFILE

This function is issued by 1DJ (print package) or 1TD (tape dump package) when a job's output is being formatted. Its purpose is to remove all dayfile information from the buffer to the disk so that only the disk need be read when a job's dayfile is to be printed. The complete dayfile flag and 'dump phase one' flag are set. If the 'complete dayfile' flag is found to be set, then this is the second time through so it is cleared along with the output register.

Only the two MTR functions issue dayfile and complete dayfile, may set phase one dump flag.

## MTR - CLOSE OUT

The dayfile buffer is dumped into the disk whenever a full sector of data is built up or whenever a job is to be printed. This process involves several steps, each of which set a flag for the subsequent phase. No entry may be made to the buffer when a dump flag is set.

On MTR's main loop a check is made to see if a dump flag is set. This flag is an address of the next phase and each phase is entered by a return jump. Every disk positioning request constitutes a different phase so that time is not wasted waiting on the disk.

Phase one requests channel 0 for the disk and phase two dump flag is set. MTR regains control and will continue its processing until the dump flag is checked again. This time phase two is entered via a return jump. If channel 0 is ready for use, a request for disk positioning to the proper track is issued and phase three dump flag is set. The current track and sector to be used by the dayfile is maintained by absolute coding. The 'update control byte' routines set the value of the current track and sector into the different dump phase locations directly.

Phase three checks channel 0 disk file status. The next sector must correspond to that set by 'update control byte' or an exit is made. One sector is written on the disk and the buffer parameters are updated accordingly. Then phase two flag is set. The buffer is dumped one sector at a time until a short sector is encountered. It is written on the disk but neither the buffer parameters nor the sector number are advanced. This scheme is used in order to maintain the dayfile as one record but still have all the information on the disk. Channel 0 is released via the output register and phase six flag is set if a spare track is assigned. If no spare track has been assigned, channel 0 is still released but phase four dump flag is set.

Phase six makes sure that the channel is released and clears the dump flag. This terminates dumping the dayfile buffer onto the disk so that normal processing may continue.

Phase four requests a track of MTR and sets phase five dump flag. When phase five is entered via a return jump, the spare track number is retrieved from the first byte of the message buffer and then the dump flag is cleared. This also completes the dayfile dump.

JOB DAYFILE LISTING

At the end of each job's output a complete history of each run during one dead start period is printed. 1DJ (print package) or 1TD (tape dump package) requests MTR to dump the dayfile buffer contents on the disk in the manner just previously described. Next, one sector of the dayfile is read and it is searched for the job's entries by 2SD (search dayfile).

Since the time a message is issued appears in the word before the job name, every word of the sector is checked for the proper job name. If the word does not match, it is copied into the peripheral buffer but its parameters are not advanced. When the name finally matches, the time has already been copied into the peripheral buffer so the job name is added in the next word. Then the subsequent message is transferred until the zero byte is encountered.

Control fluctuates between the dump package, i.e. 1DJ or 1TD, which reads a sector of the dayfile, and 2SD, which searches it for a particular job name. The dayfile is searched in this manner until a short

sector is found. When it is encountered, a MTR function requesting assignment of PP time to the control point is made. This computes the total PP running time and stores it in word 24 of the control point area. 2SD converts this time to decimal seconds and then sends out a dayfile message "PP XXXX SEC".

A top of form request is made as the first entry into a circular buffer in central memory. The peripheral buffer containing the dayfile information for this job is copied to the circular buffer. An entry of the same type, "PP XXXX SEC", that was sent to the dayfile is added to the circular buffer. Now the job's dayfile is complete and ready for printing.

<u>NOTES</u>

1. The dayfile is the first entry in FNT. It is set from the library tape and is of common type so that any program may access it.

2. Any message sent to the dayfile also appears as a console message (line 3 of the control point display)

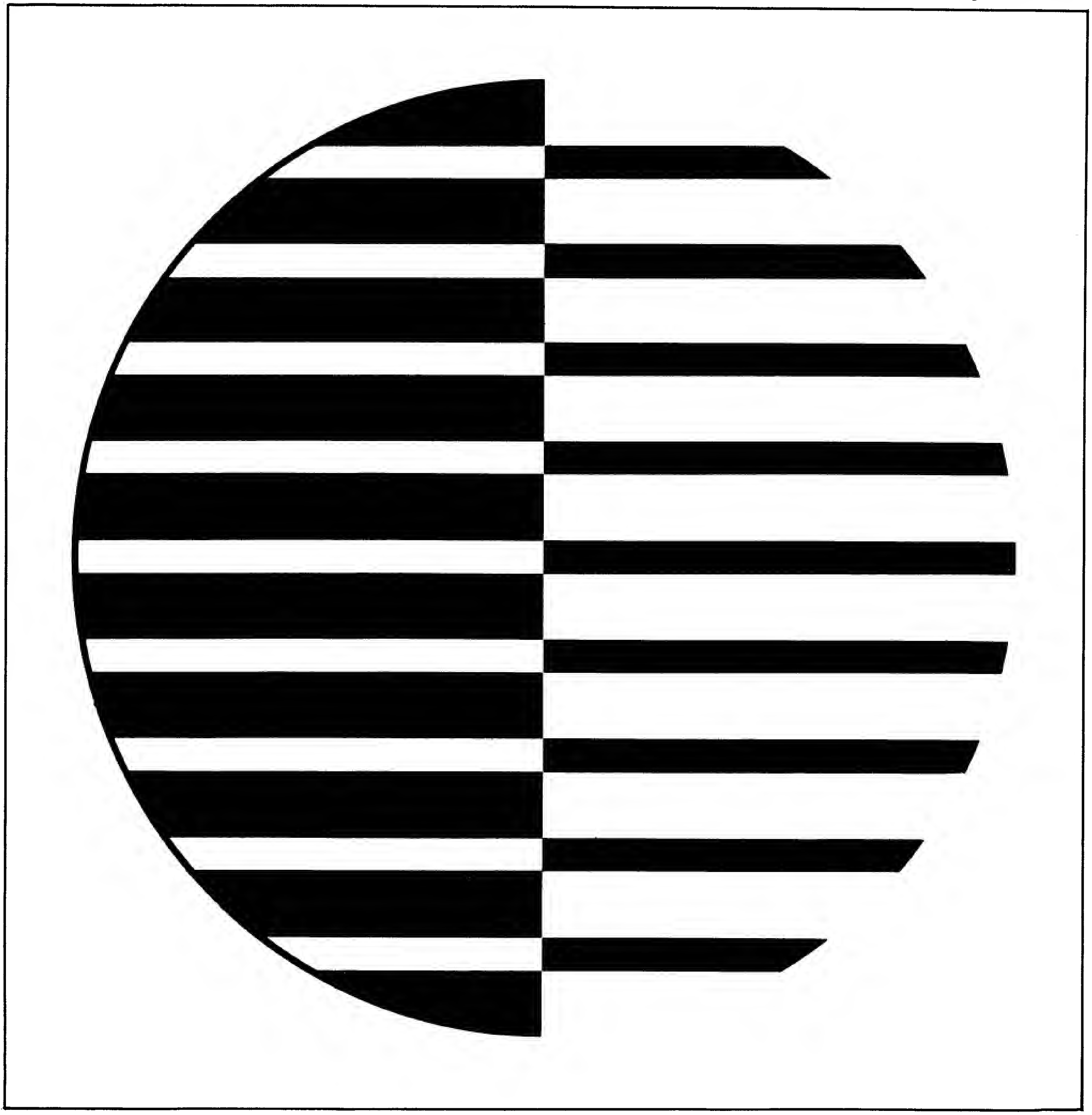
CONTROL DATA
CORPORATION 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

PRINTED IN U.S.A.

# CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

## CHIPPEWA OPERATING SYSTEM DOCUMENTATION

Volume III    Preliminary Edition



CONTROL DATA CORPORATION

Development Division - Applications

# DSD - THE SYSTEM DISPLAY

Chippewa Operating System

# DSD: The System Display

## Introduction

The display console is controlled by a display program, DSD, which permanently resides in peripheral processor 9. DSD displays a variety of information concerning the status of the system, including a display of the dayfile, a display of the jobs waiting to be executed and waiting to be printed, and a display showing the status of each control point. DSD permits selected portions of central memory to be displayed, and also provides for the modification of central memory locations. In addition to its display function, DSD processes keyboard messages from the operator. Operator functions include bringing a job to or dropping a job from a control point, assigning equipment, and selection of various types of displays. The Chippewa Operating System also contains a job display package, DIS. DIS, when called, is assigned to a control point, and permits the modification of job parameters, memory locations, and control statement sequences for this job assigned to the control point. For the most part, DSD and DIS displays are identical.

The main components of the display console are the two cathode ray tubes and the keyboard. By issuing the appropriate function codes to the display console controller, displays of 16, 32, or 64 characters per line may be selected on either the right or left screens. A dot mode display is also available, although only the character mode display is used by the operating system. The display area can be considered to be composed of a grid of points, 512 by 512 points in size. A display can be initiated at any point in the display area by issuing the coordinates of that point. A vertical, or Y, coordinate is sent to the controller in the low-order nine bits of a byte in which the high-order octal digit is a 7. Similarly, a horizontal, or X, coordinate is sent to the controller in the low-order nine bits of a byte in which the high-order octal digit is a 6. If the display console controller receives a byte in which the high-order octal digit is neither a 6 or a 7, it is assumed that this byte contains two display code characters.

To display a line of information on the screen, an X and a Y coordinate are sent to the controller via the appropriate output instructions

(OAN or OAM). These coordinates define the location of the lower left corner of the first character to be displayed. The information to be displayed is then sent to the controller via an OAM instruction. As each character is displayed the X coordinate is automatically incremented. To display another line, the X and Y coordinates should be reinitialized. A coordinate of X=000 defines the left-most boundary of the display area; a coordinate of X=$777_8$ defines the right-most boundary of the display area; a coordinate of Y=$777_8$ defines the upper boundary of the display area, and a coordinate of Y=000 defines the lower boundary of the display area.

The Chippewa Operating System uses a display of 64 characters per line in both DSD and DIS. Generally, the Y (vertical) coordinate spacing between successive lines is $12_8$. The display must be regenerated at least 25 times per second in order to avoid flicker. The DSD display is designed to maintain an average rate of 40 displays per second.

DSD Master Loop

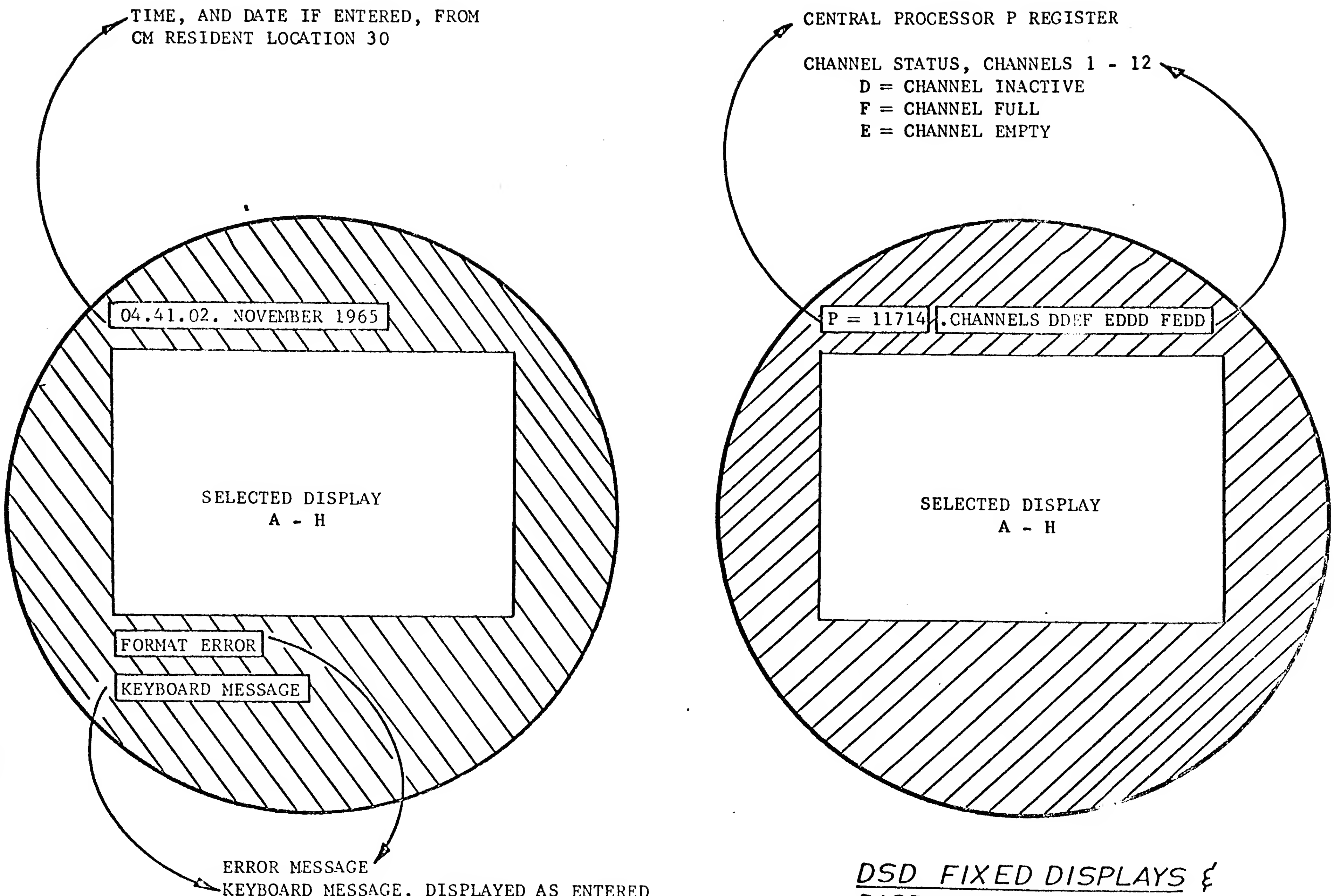
The DSD master loop is shown on page A-1 of the attached flow charts. On the initial entry to this routine (i.e., at dead start time), a subroutine is called to perform housekeeping. This subroutine clears the temporary storage areas used by DSD, selects the "A" display (dayfile) on the left screen and the "B" display (control points) on the right screen, and requests reservation of channel 10 from MTR. Display selection in DSD is performed by setting the address of the desired display subroutine in location 70 for the left screen and in location 71 for the right screen.

On each pass through its master loop, DSD selects the display console keyboard and issues an input instruction to read the keyboard. If a zero byte is returned, then no key has been depressed since the last pass through the master loop. If a non-zero byte is returned, the keyboard character in the low-order bits of the byte is processed. If the character is a carriage return, the Message Ready flag is set and the keyboard message is processed.

After keyboard processing is completed, DSD issues a function code to select the left screen, and displays the time and date from central memory resident beginning at location 30. The contents of this area are read and displayed, word by word, until a zero byte is encountered. Regardless of the display selected for the left screen, the time-date line is always displayed. DSD then jumps to the subroutine whose address is contained in location 70 to process the selected left screen display. At the bottom of the left screen, the keyboard message currently being entered is displayed. If an error is encountered in processing this message, the error message "FORMAT ERROR" will be displayed immediately above the keyboard message. Once processing of a valid message is complete, the message will be no longer displayed.

DSD then issues a function code to display the right screen. At the top of the right screen, the contents of the central processor P register and the status of the 12 data channels are always displayed. DSD reads the central processor P register, converts the contents of the P register to display code, and displays these characters. On the same line, three groups of four characters, one character for each of the 12 data channels, are displayed. Each channel is first tested to determine if it is active or inactive. If the channel is inactive, the displayed character corresponding to that channel is a "D" (disconnected). If the channel is active and empty, an "E" is displayed, while if the channel is active and full, an "F" is displayed. (See figure 1) DSD then jumps to the subroutine whose address is contained in location 71 to process the selected right screen display.

After both screens have been displayed, DSD calls the Adjust Display Period subroutine (shown on page A-1 of the attached flow charts). The purpose of this subroutine is to control the number of passes made through DSD's master loop in a fixed time interval in order to avoid flicker. On each entry to the Adjust Display Period subroutine (i.e., on each pass through DSD's master loop), a display cycle counter is advanced. At the end of each second, the display cycle count is examined to determine if the display was repeated more than $50_8$ times in the past second. If it was, a delay count (D) equal to the display cycle count - $50_8$ is set. If the display was repeated less than 50 times in the past second, this delay count is set to zero. The delay count is used to establish




DSD FIXED DISPLAYS &
DISPLAY PLACEMENT

Figure 1

a delay between successive passes through DSD's master loop: the larger the delay, the greater the time between successive passes through the master loop. Also, in the next second, D will be set to zero if the display is repeated 50-D times or less. If in any second, then, the display is repeated more than $50_8$ times, a delay so that the display will be repeated less than $50_8$ times in the subsequent second: over a period of several seconds, the display rate should average out to $50_8$ times per second. At the end of each second, the display cycle counter is reset to zero.

In a 6000 system with a single display console, DSD must relinquish control to DIS when the latter is called to a control point. When DIS begins execution, it requests MTR to assign an equipment of type DS. MTR searches the EST for an entry of this type, and, when found, enters the requestor's control point address in byte one of the EST entry. On each pass through its master loop, DSD reads the EST entry for equipment number 10. If byte one is non-zero, then this equipment has been assigned to another user - DIS. DSD then releases the channel reservation for the channel to which the display controller is connected, and loops on a test of byte one of the EST entry. When this byte becomes zero once again, DIS has released control of the console, and so DSD requests the channel once again and returns to its master loop.

One of the keyboard entries processed by DSD is the "STEP." message, which causes MTR to enter a step mode of operation. In step mode, MTR pauses for operation intervention before processing each request from a peripheral processor. To process the STEP message, DSD sends function request 5 to MTR. MTR then sets a switch in the subroutine which processes requests from peripheral processors. When a request is next received from a peripheral processor, MTR will set a Wait flag in byte 5 of central memory location 14, and will then loop until this flag has been modified. Entering a space on the keyboard will result in the clearing of this flag by DSD: MTR will then process the request just received, but will pause again before processing subsequent requests. Entering a period on the keyboard will result in this flag's being set to 7777 by DSD: MTR will then reset the switch in the subroutine which processes peripheral processor requests, and will process subsequent requests in the normal manner. Since a space or a period is not, in

itself, a conventional DSD message, DSD checks for the entry of these characters on each pass through its master loop if MTR is in step mode.

DSD Keyboard Message Processing

The processing of characters received from the keyboard is shown in the flow chart on page A-1. If the character received is a carriage return, then a complete message has been entered and so the Message Ready flag is set. If the character received is a backspace, DSD clears the last character entered in the buffer, resets the buffer address accordingly, and clears the error flag which may have been set if an attempt was made to process the message. If the drop key was depressed, then the entire message is deleted: the buffer address is reset to the starting address, and the error flag cleared. Should the character be a valid keyboard character, it is entered in the message buffer and the buffer address advanced. Note that the space character from the keyboard ($62_8$) is not a display code character, and so a blank ($55_8$) is substituted for it.

When DSD detects that a carriage return has been entered on the keyboard, the Message Ready flag is set to indicate that a message is ready for processing. DSD then proceeds to interpret the message. Message processing is illustrated in the flow chart on page A-2. The second character is examined to determine if it is a period: if it is, then the message is a control point message, and so the first character is examined to determine if it is a valid control point number (1-7). If the first character is not a numeric in the range 1-7, the Message Error flag is set and control returned to DSD's master loop, where the message "FORMAT ERROR" will be displayed. If the first character is a valid control point number, then a table search is made for the address of the appropriate subroutine. If the message is not found in the table, the Message Error flag is set and control is returned to DSD's master loop. Processing of the valid control point messages is described below.

ONSW: If the message is of the form n.ONSWx., DSD sets the bit corresponding to X+5 in byte 5 of location RA for control point n, and in word 26 of control point area n. Control is then returned to DSD's master loop.

OFFSW: If the message is of the form n.OFFSWx., DSD clears the bit corresponding to X+5 in byte 5 of location RA for control point n, and in word 26 of control point area n. Control is then returned to DSD's master loop.

LOAD: If the message is of the form n.LOAD., DSD writes the package name (1LT) and the control point number, n, in its Message Buffer. Word 21 of control point area n is then examined to determine if the control point area contains a job name. If it does not, DSD requests MTR to assign a pool processor to control point n. MTR will copy the contents of DSD's Message Buffer into the Input Register of a free pool processor, and assign the processor to control point n. Control is then returned to DSD's master loop. If the control point area contains a job name (word 21 non-zero), control is returned to the DSD master loop. If the control point area contains a job name (word 21 non-zero), control is returned to the DSD master loop without requesting the assignment of a processor.

NEXT: Processing of the message n.NEXT. is identical to the processing of the LOAD message with the exception that the package name 1BJ is written in DSD's Message Buffer.

READ: Processing of the message n.READ. is identical to the processing of the LOAD message with the exception that the package name 1LJ is written in DSD's Message Buffer.

PRINT: Processing of the message n.PRINT. is identical to the processing of the LOAD message with the exception that the package name 1DJ is written in DSD's Message Buffer.

DIS: If the message is of the form n.DIS., DSD writes the package name DIS in its Message Buffer, and requests MTR to assign a pool processor to control point n. Control is then returned to DSD's master loop.

ASSIGN: The ASSIGN message is generally entered in response to a REQUEST statement display or the message WAITING FOR XX. If the message is of the form n.ASSIGNXX., DSD requests MTR to assign the

specified equipment to the control point. MTR looks up the corresponding entry in the EST: if equipment XX is not already assigned, then MTR assigns the equipment to control point n and writes the equipment number in word 22 in the control point area. After initiating the MTR request, control is returned to DSD's master loop.

GO: The $2^0$ bit in byte 4 of location RA is a pause bit. This bit is set by a FORTRAN PAUSE statement, and is also set by certain peripheral packages when an error is detected. For example, 2RT sets this bit when a parity error has occurred after reading a record three times. When the n.GO. statement is processed, this bit is cleared. Also, the most recent message in the control point area (presumably the PAUSE statement) is cleared.

END: If the message is of the form n.END1., n.END2., n.END3., or n.END4., DSD sets a printer stop code in byte 2 of word 20 in the control point area. This stop code is equivalent to setting the low-order second octal digit of this byte to the digit following the word END in the message. The printer stop code is sensed by the four-printer print programs.

DROP: If the message is of the form n.DROP., DSD writes the control point number n in its Output Register, and requests MTR to drop the job at control point n. MTR sets error flag six (Operator Drop) to initiate error processing.

If the second character in the message was a period, and the message was not found to be one of those described above, the Message Error flag is set and control returned to DSD's master loop. If the second character in the message was not a period, the first character is examined to determine if it is an octal digit. Should the first character be an octal digit, it is assumed that the message is a storage entry message of the form a,d., where a represents a central memory address and d represents the data to be entered in memory at that address. The characters in the message are assembled and converted to octal until a separator is found: if the separator is not a comma, the Message Error flag is set. Once the address has been assembled, the characters following the comma

are assembled and converted to octal until another separator is found. If this separator is not a period, the Message Error flag is set. The assembled data is stored, right-justified, in a 5-byte area, and the contents of this area are then written in central memory at the specified address.

If the first character is not an octal digit, the third character is examined to determine if it is a period. If the third character is a period, the message is assumed to be a display mode message of the form AB., where A and B represent characters specifying the desired display on the left and right screens respectively. The subroutine address corresponding to the specified display (A-H) is located in a table and stored in location 70, in the case of the left screen display, or location 71, in the case of the right screen display.

If the third character was not a period but was a comma, it is assumed that the message is a display field change message of the form mf,a., where m is the display mode (C-G), A is the field whose starting address is to be changed (0-3 for fields 0-3, or 4 for all four fields), and a is the new starting address. Each of the storage display subroutines for storage displays C, D, E, F, and G maintains a list of four addresses, one for each of the four fields displayed. When this message is detected, DSD modifies the appropriate address in the list of field addresses for the specified display if the second character is 0-3. If the second character of the message is 4, the first address in the list is set to the address contained in the message, the second address in the list is set to the address contained in the message plus $10_8$, and so forth.

If the third character in the message is not a comma, it is assumed to be a non-control point message of the form described below. DSD searches a table for the address of the appropriate subroutine: if the message is not found in the table, the Message Error flag is set and control returned to DSD's master loop. Processing of these messages is described below.

<u>DCN</u>: If the message is of the form DCNXX., where XX is an octal channel number, DSD assembles the channel number and tests the channel to determine if it is inactive. If the channel is active,

a channel disconnect is issued.

FCN: If the message is of the form FCNXX., where X is an octal channel number, the channel number is assembled and a test made to determine if the channel is inactive. If the channel is inactive, a zero function is sent to the channel.

AUTO: If the message is AUTO., DSD assigns READ (1LJ) to control point 1, PRINT (1DJ) to control point 2, and NEXT (1BJ) to control points 3, 4, 5, and 6. To assign a package to a control point, DSD writes the package name and control point number in bytes one and two of word one of its Message Buffer, and sends function request 20, Assign PPU, to MTR. MTR locates a free pool processor, assigns it to the control point, and copies word one of DSD's Message Buffer into the pool processor's Input Register.

STEP: If the message is STEP., DSD requests MTR to enter step mode by sending function request 5 to MTR. (See discussion on page 5.)

ON or OFF: The messages ONXX. and OFFXX. permit the operator to clear and set, respectively, the interlock bit in the EST table entries. In these messages, XX is an octal equipment number which defines a location in EST. The interlock bit is generally used only with magnetic tape units. When this bit is set, the corresponding equipment will not be automatically allocated in response to an ASSIGN request: if this bit is cleared, and a request such as ASSIGN MT is processed, the equipment will be automatically assigned by MTR. For equipment types MT and WT, this bit is set at load time. (See page 9 of CENTRAL MEMORY RESIDENT.)

TIME: If the message begins with the characters TIME., and contents of the keyboard message buffer following these characters are copied into central memory resident beginning at location 30. This information may comprise up to six central memory words. It is assumed that the first portion of this message has the form _HR.MN. SC., where HR represents hours, MN represents minutes, and SC represents seconds. This time will be advanced by MTR and will

appear in all dayfile messages and at the top of the left screen display. The information following the time may be the date and/or any other desired information. The date portion will also be displayed at the top of the left screen, and will be printed at the end of a job's dayfile listing.

DSD Displays

The various display modes which may be selected in DSD are as follows:

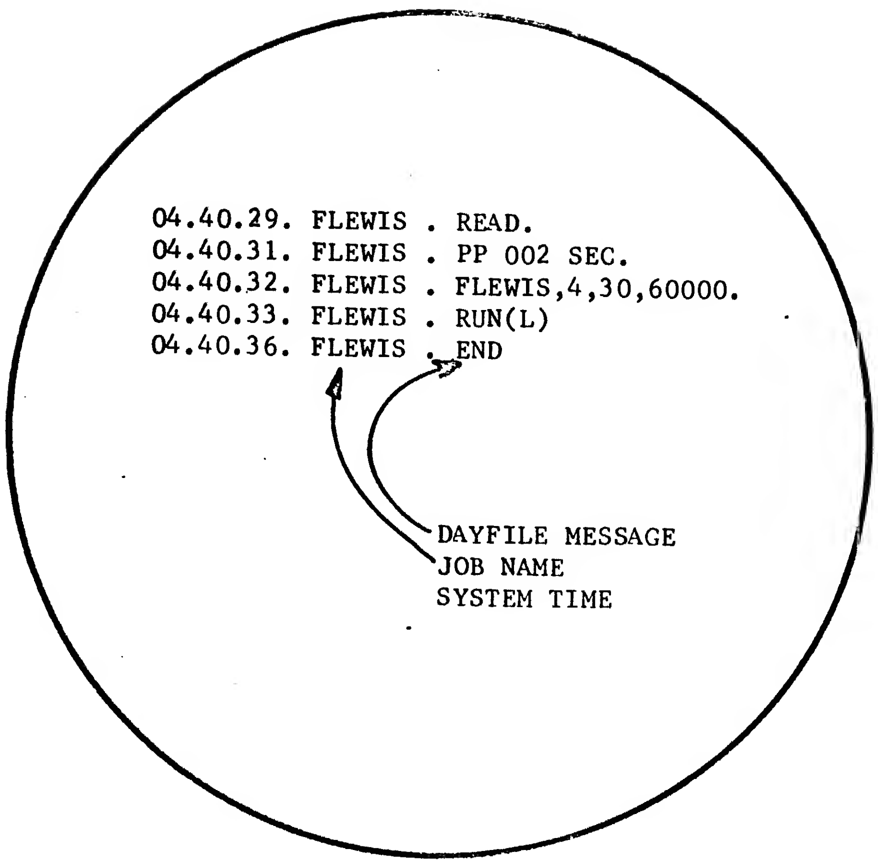
A.........Dayfile Display
B.........Control Point Display
C.........Storage Display (5 groups of 4 digits)
D.........Storage Display (5 groups of 4 digits)
E.........Storage Display (5 groups of 4 digits)
F.........Storage Display (4 groups of 5 digits)
G.........Storage Display (4 groups of 5 digits)
H.........Job Backlog Display

The format of these displays is illustrated in figures 2, 3, and 4. Each display is processed by a separate DSD subroutine: these subroutines appear on pages A-3 and A-4 of the attached flow charts. For the most part, display processing is quite straightforward, and discussions of these displays will be limited to points of interest.

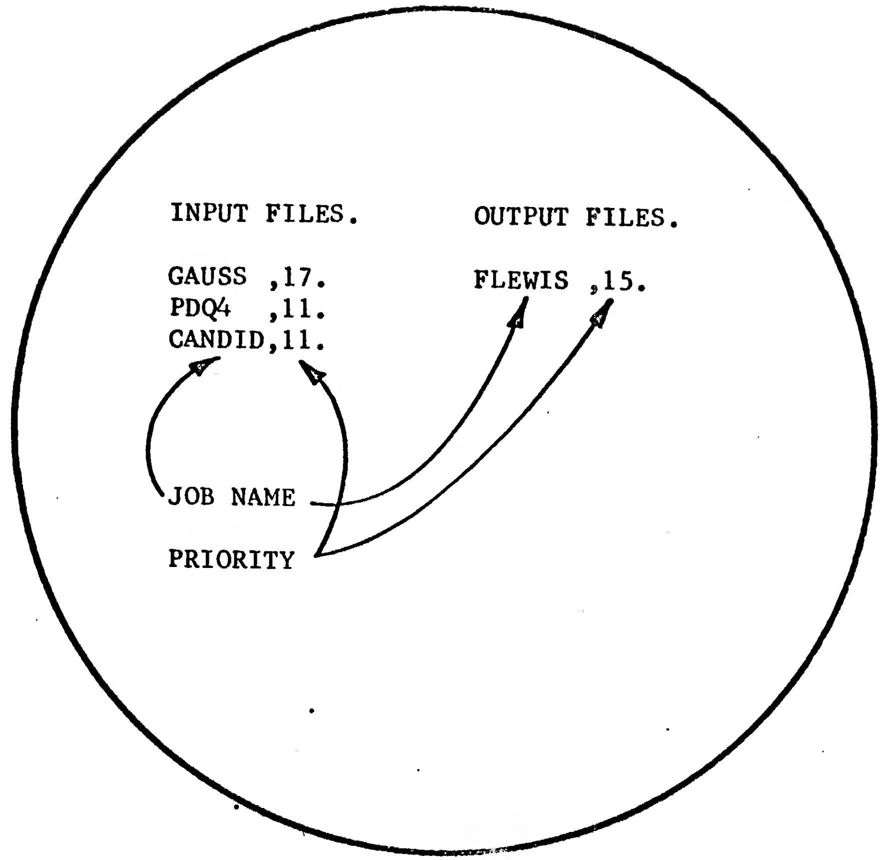
"A" Display: The "A" display is a display of the dayfile buffer (DFB) contents from FIRST to IN. It is possible that the bottom of the display area may be reached before all the information in the dayfile has been displayed. If so, the subroutine parameters are modified so that on the next entry to the subroutine, the message which previously appeared at the top of the display will not be processed, thus permitting a new message to be displayed. Also, the point at which the display begins is moved down by the width of a line, and gradually moved back up during the next 10 displays. As a result, a revolving or rolling effect is obtained.

"B" Display: The B display shows information concerning each of the seven control points, as shown in figure 3. On entry to this subroutine, the copy of the control stack which MTR maintains in locations 56-57 of central memory resident is read. For each control

Figure 2




DISPLAY "A" - DAYFILE




DISPLAY "H" - JOB BACKLOG

DSD A & H DISPLAYS

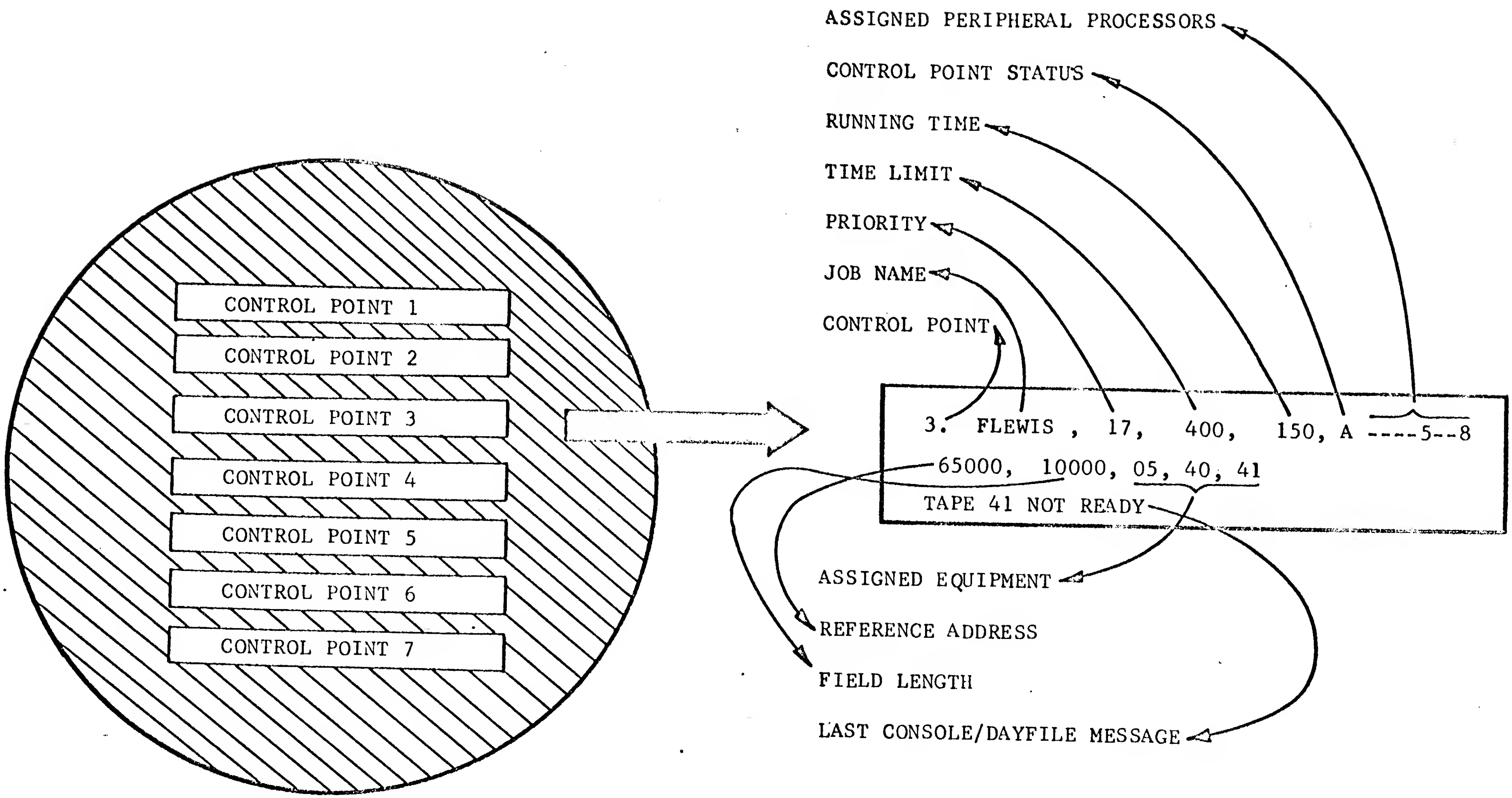



Figure 3

DSD "B" DISPLAY: CONTROL POINTS

point in the stack, a status indicator, A-G, is set to represent the position of that control point in the stack (i.e., A represents the top of the stack, B the second entry in the stack, and so forth). The status byte in each control point is also read, and the status indicator set to W or X depending on the setting of these flags. The remaining processing performed by the subroutine consists of reading information from the control point area and displaying this information.

<u>Displays C-G</u>: The storage displays, C through G, each display 4 fields of 8 central memory words (see figure 4). Displays C, D, E are identical in format and display each central memory word as 5 groups of 4 octal digits. Displays F and G are identical in format and display each central memory word as 4 groups of 5 octal digits. There is a separate subroutine for each of these five displays: each of these subroutines maintains a list of four field addresses which specify the starting point for each of the four 8-word field displays. These address lists are set via keyboard messages (see page 9). The address lists are initialized at load time as follows:

C Display .....Words 20-27 of control point areas 1, 2, 3, and 4
D Display .....CM resident locations 0-37
E Display .....CM resident locations 60-117
F Display .....Central Memory locations 10000-10037
G Display .....Central Memory locations 10040-10077

The reason for identical displays (C, D, E, and F, G) is to permit the operation to switch between scans of selected memory areas without the necessity of entering starting addresses each time he switches from one area to another.

<u>"H" Display</u>: The H display lists the input and output files in the FNT. Upon entry to the H display subroutine, the FNT is searched and two lists prepared: one, a list of FNT addresses for entries of file type INPUT, and the other a list of FNT addresses for entries of file type OUTPUT. The entries represented in these lists

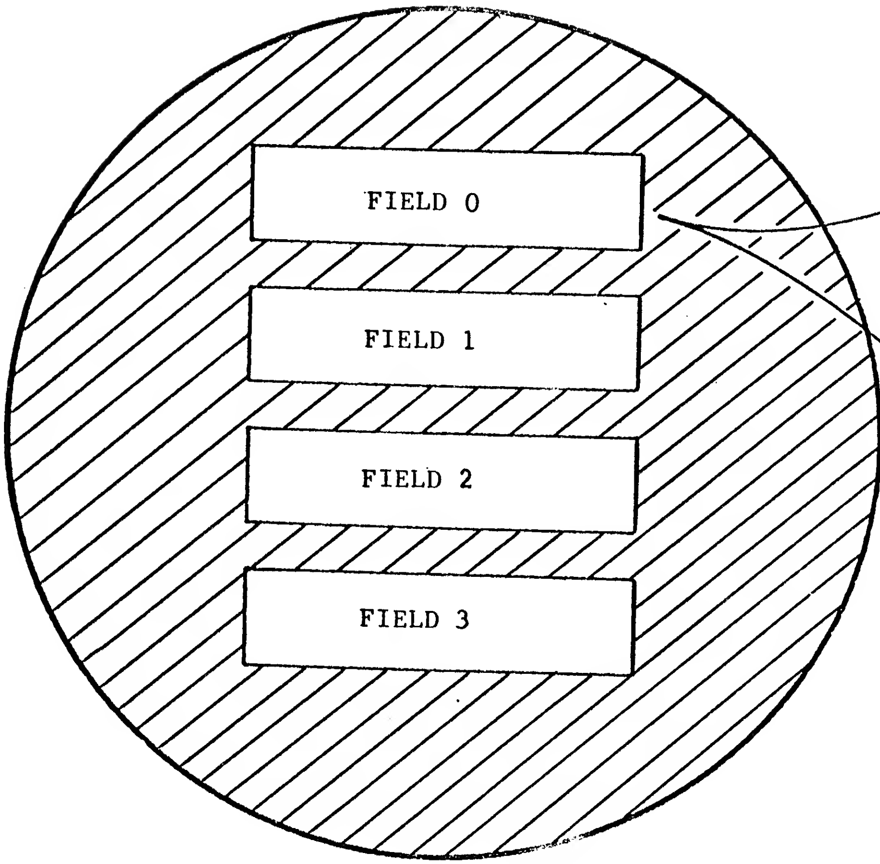
Figure 4

CONTENTS OF LOCATION

LOCATION

| | | | | |
|---|---|---|---|---|
| 04600 | 00000 | 00000 | 00000 | 00000 |
| 04601 | 00000 | 00000 | 00000 | 01000 |
| 04602 | 51202 | 60002 | 63220 | 43052 |
| 04603 | 67312 | 51300 | 00002 | 11730 |
| 04604 | 56730 | 51402 | 60003 | 10740 |
| 04605 | 51700 | 00002 | 71730 | 00003 |
| 04606 | 76710 | 51730 | 00004 | 67112 |
| 04607 | 61102 | 60005 | 01004 | 00200 |

4 GROUPS OF 5 OCTAL DIGITS
(DISPLAYS F AND G)

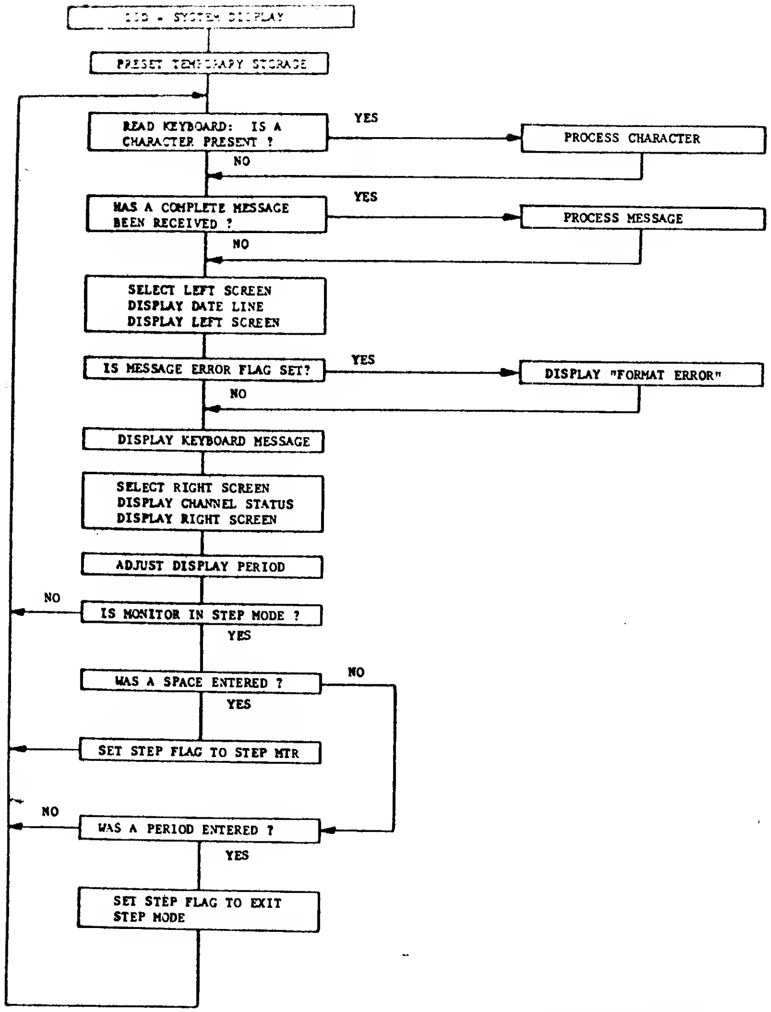


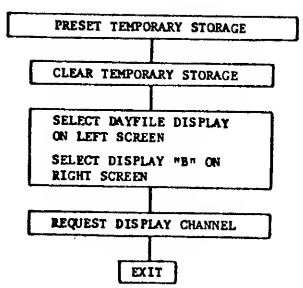

| | | | | | |
|---|---|---|---|---|---|
| 04600 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 04601 | 0000 | 0000 | 0000 | 0000 | 1000 |
| 04602 | 5120 | 2600 | 0263 | 2204 | 3052 |
| 04603 | 6731 | 2513 | 0000 | 0021 | 1730 |
| 04604 | 5673 | 0514 | 0260 | 0031 | 0740 |
| 04605 | 5170 | 0000 | 0271 | 7300 | 0003 |
| 04606 | 7671 | 0517 | 3000 | 0046 | 7112 |
| 04607 | 6110 | 2600 | 0501 | 0040 | 0200 |

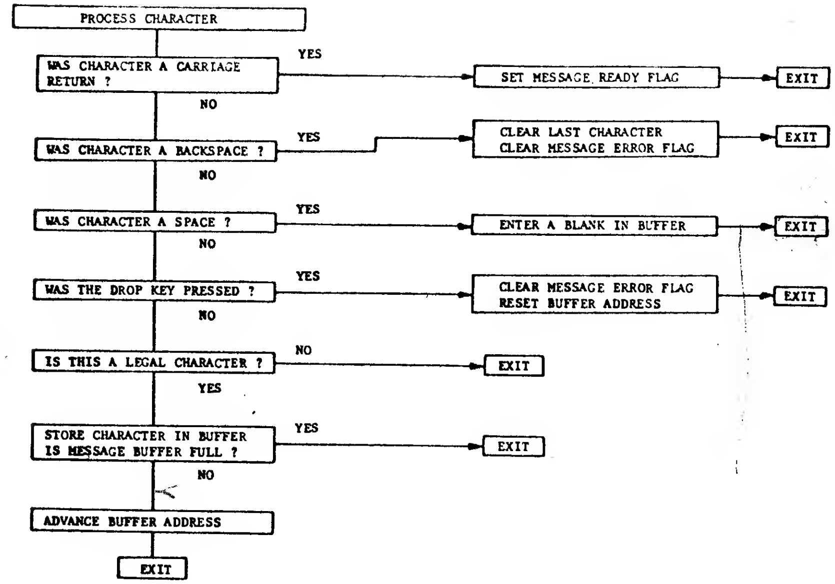
5 GROUPS OF 4 OCTAL DIGITS
(DISPLAYS C, D, AND E)

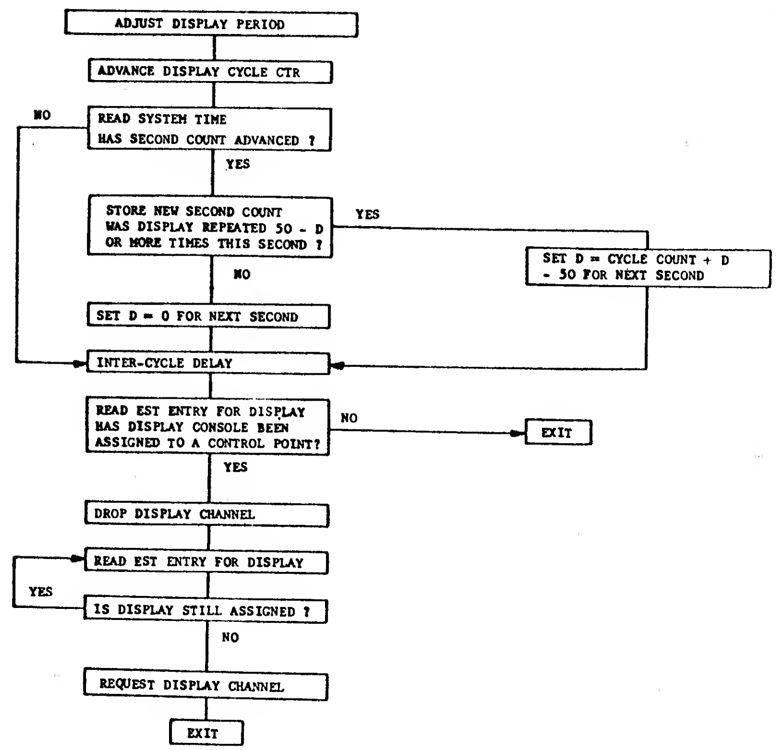
DSD DISPLAYS C THRU G

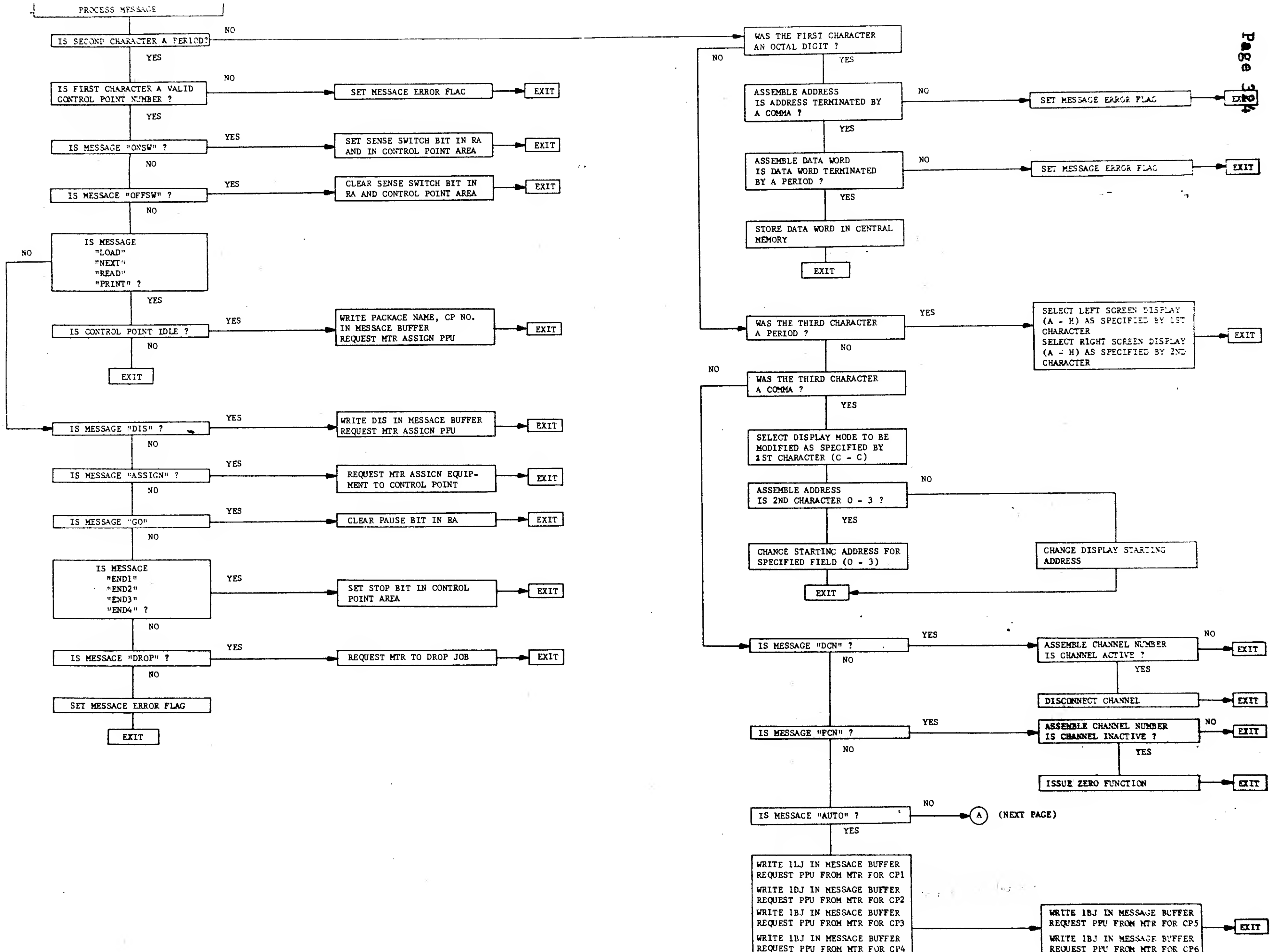
are then read, and the file name and priority displayed. These lists are updated only at intervals of 1/10 of a second in order to reduce unnecessary read pyramid conflicts.


DSD - SYSTEM DISPLAY
PRESET TEMPORARY STORAGE
READ KEYBOARD: IS A CHARACTER PRESENT ?
YES
PROCESS CHARACTER
NO
HAS A COMPLETE MESSAGE BEEN RECEIVED ?
YES
PROCESS MESSAGE
NO
SELECT LEFT SCREEN
DISPLAY DATE LINE
DISPLAY LEFT SCREEN
IS MESSAGE ERROR FLAG SET?
YES
DISPLAY "FORMAT ERROR"
NO
DISPLAY KEYBOARD MESSAGE
SELECT RIGHT SCREEN
DISPLAY CHANNEL STATUS
DISPLAY RIGHT SCREEN
ADJUST DISPLAY PERIOD
NO
IS MONITOR IN STEP MODE ?
YES
WAS A SPACE ENTERED ?
NO
YES
SET STEP FLAG TO STEP MTR
NO
WAS A PERIOD ENTERED ?
YES
SET STEP FLAG TO EXIT STEP MODE


PRESET TEMPORARY STORAGE
CLEAR TEMPORARY STORAGE
SELECT DAYFILE DISPLAY ON LEFT SCREEN
SELECT DISPLAY "B" ON RIGHT SCREEN
REQUEST DISPLAY CHANNEL
EXIT

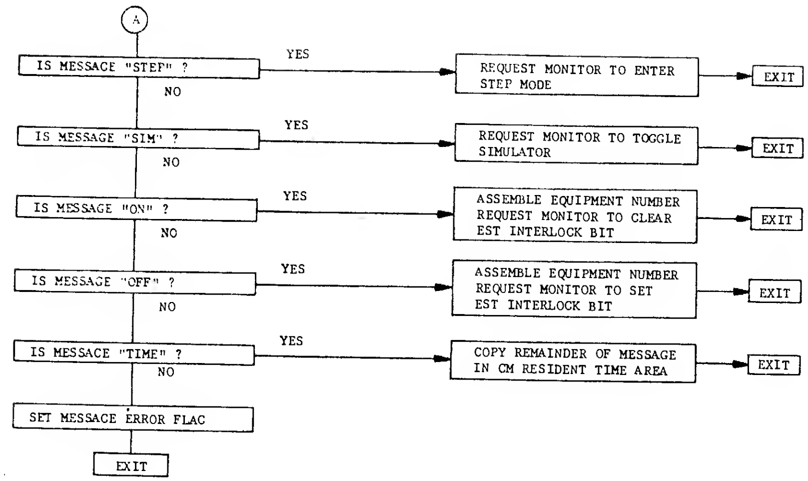

PROCESS CHARACTER
WAS CHARACTER A CARRIAGE RETURN ?
YES
SET MESSAGE READY FLAG
EXIT
NO
WAS CHARACTER A BACKSPACE ?
YES
CLEAR LAST CHARACTER
CLEAR MESSAGE ERROR FLAG
EXIT
NO
WAS CHARACTER A SPACE ?
YES
ENTER A BLANK IN BUFFER
EXIT
NO
WAS THE DROP KEY PRESSED ?
YES
CLEAR MESSAGE ERROR FLAG
RESET BUFFER ADDRESS
EXIT
NO
IS THIS A LEGAL CHARACTER ?
NO
EXIT
YES
STORE CHARACTER IN BUFFER
IS MESSAGE BUFFER FULL ?
YES
EXIT
NO
ADVANCE BUFFER ADDRESS
EXIT

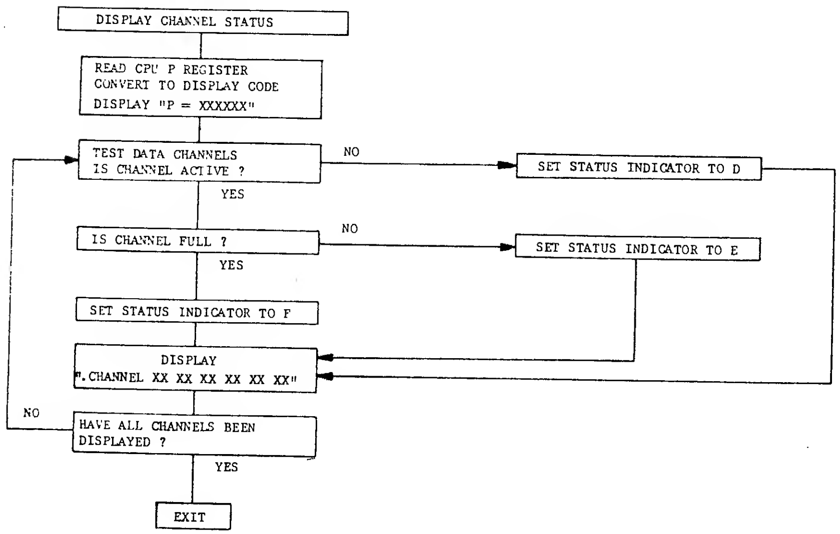

ADJUST DISPLAY PERIOD
ADVANCE DISPLAY CYCLE CTR
NO
READ SYSTEM TIME
HAS SECOND COUNT ADVANCED ?
YES
STORE NEW SECOND COUNT
WAS DISPLAY REPEATED 50 - D OR MORE TIMES THIS SECOND ?
YES
SET D = CYCLE COUNT + D - 50 FOR NEXT SECOND
NO
SET D = 0 FOR NEXT SECOND
INTER-CYCLE DELAY
READ EST ENTRY FOR DISPLAY
HAS DISPLAY CONSOLE BEEN ASSIGNED TO A CONTROL POINT?
NO
EXIT
YES
DROP DISPLAY CHANNEL
READ EST ENTRY FOR DISPLAY
YES
IS DISPLAY STILL ASSIGNED ?
NO
REQUEST DISPLAY CHANNEL
EXIT

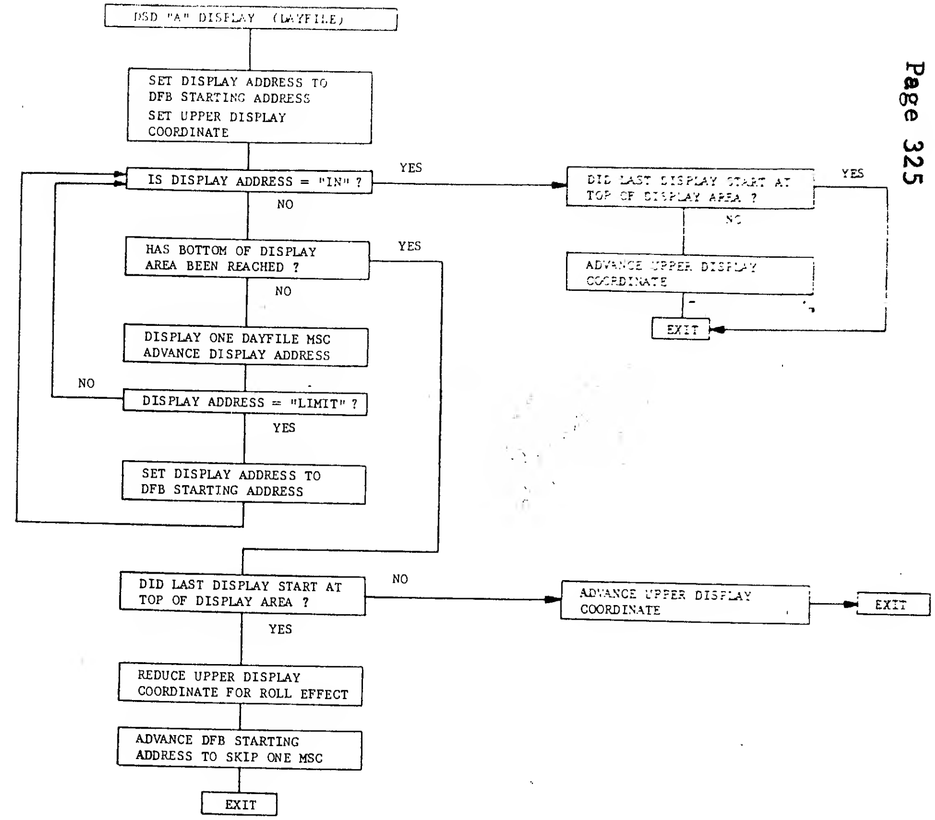

PROCESS MESSAGE
IS SECOND CHARACTER A PERIOD?
NO
YES
IS FIRST CHARACTER A VALID CONTROL POINT NUMBER ?
SET MESSAGE ERROR FLAG
EXIT
IS MESSAGE "ONSW" ?
SET SENSE SWITCH BIT IN RA AND IN CONTROL POINT AREA
IS MESSAGE "OFFSW" ?
CLEAR SENSE SWITCH BIT IN RA AND CONTROL POINT AREA
IS MESSAGE "LOAD" "NEXT" "READ" "PRINT" ?
IS CONTROL POINT IDLE ?
WRITE PACKAGE NAME, CP NO. IN MESSAGE BUFFER REQUEST MTR ASSIGN PPU
IS MESSAGE "DIS" ?
WRITE DIS IN MESSAGE BUFFER REQUEST MTR ASSIGN PPU
IS MESSAGE "ASSIGN" ?
REQUEST MTR ASSIGN EQUIPMENT TO CONTROL POINT
IS MESSAGE "GO"
CLEAR PAUSE BIT IN RA
IS MESSAGE "END1" "END2" "END3" "END4" ?
SET STOP BIT IN CONTROL POINT AREA
IS MESSAGE "DROP" ?
REQUEST MTR TO DROP JOB
SET MESSAGE ERROR FLAG
WAS THE FIRST CHARACTER AN OCTAL DIGIT ?
ASSEMBLE ADDRESS IS ADDRESS TERMINATED BY A COMMA ?
ASSEMBLE DATA WORD IS DATA WORD TERMINATED BY A PERIOD ?
STORE DATA WORD IN CENTRAL MEMORY
WAS THE THIRD CHARACTER A PERIOD ?
SELECT LEFT SCREEN DISPLAY (A - H) AS SPECIFIED BY 1ST CHARACTER SELECT RIGHT SCREEN DISPLAY (A - H) AS SPECIFIED BY 2ND CHARACTER
WAS THE THIRD CHARACTER A COMMA ?
SELECT DISPLAY MODE TO BE MODIFIED AS SPECIFIED BY 1ST CHARACTER (C - C)
ASSEMBLE ADDRESS IS 2ND CHARACTER 0 - 3 ?
CHANGE STARTING ADDRESS FOR SPECIFIED FIELD (0 - 3)
CHANGE DISPLAY STARTING ADDRESS
IS MESSAGE "DCN" ?
ASSEMBLE CHANNEL NUMBER IS CHANNEL ACTIVE ?
DISCONNECT CHANNEL
IS MESSAGE "FCN" ?
ASSEMBLE CHANNEL NUMBER IS CHANNEL INACTIVE ?
ISSUE ZERO FUNCTION
IS MESSAGE "AUTO" ?
A
(NEXT PAGE)
WRITE 1LJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP1
WRITE 1DJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP2
WRITE 1BJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP3
WRITE 1BJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP4
WRITE 1BJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP5
WRITE 1BJ IN MESSAGE BUFFER REQUEST PPU FROM MTR FOR CP6

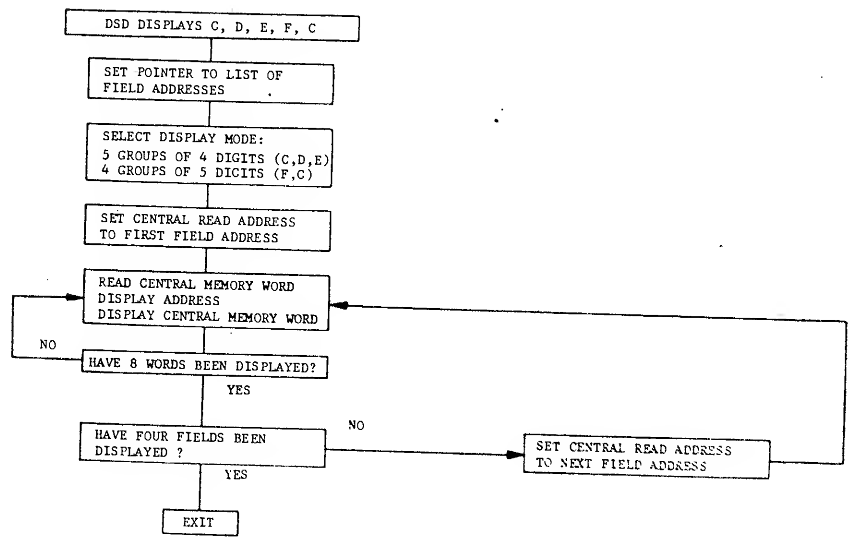
DSD: PAGE A-2

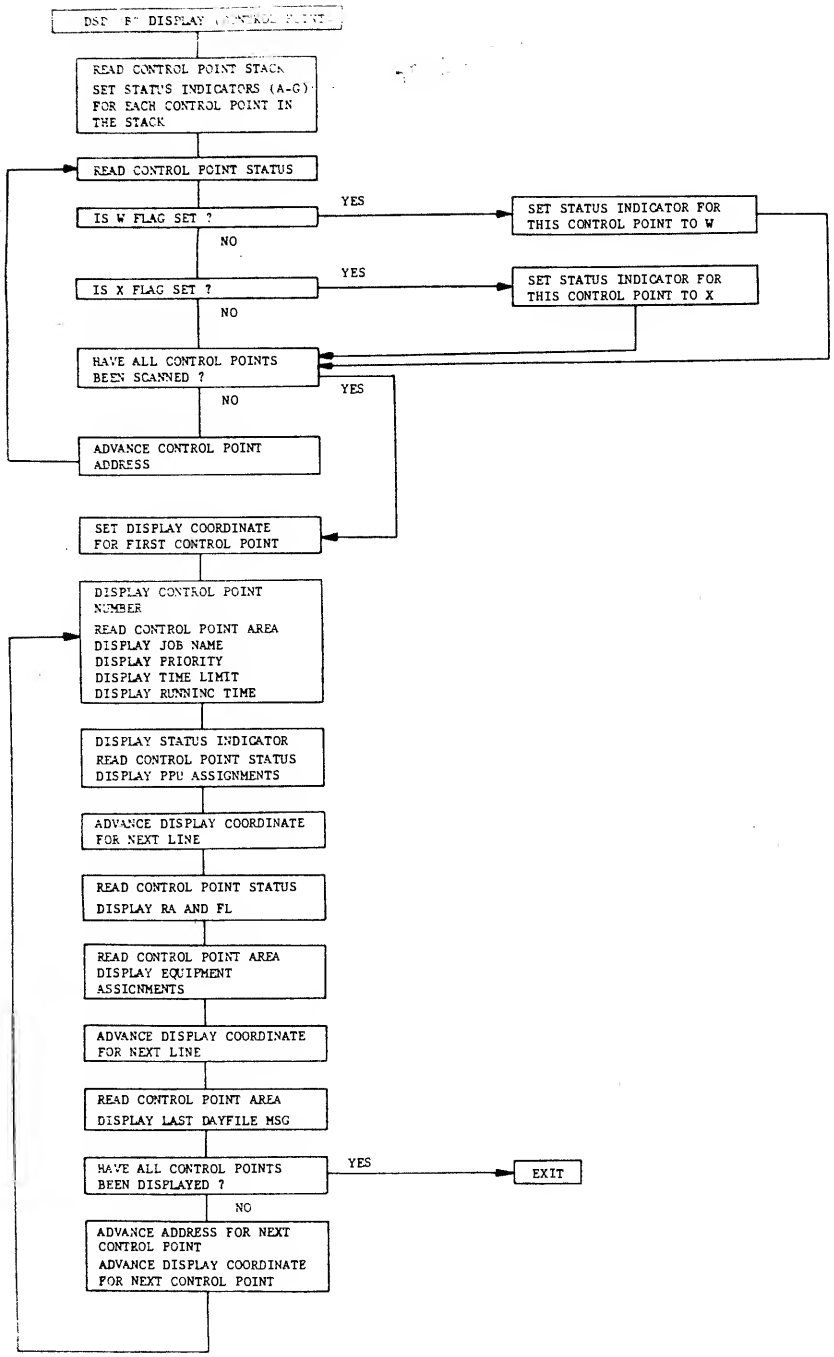
PROCESS MESSAGE, CONTINUED

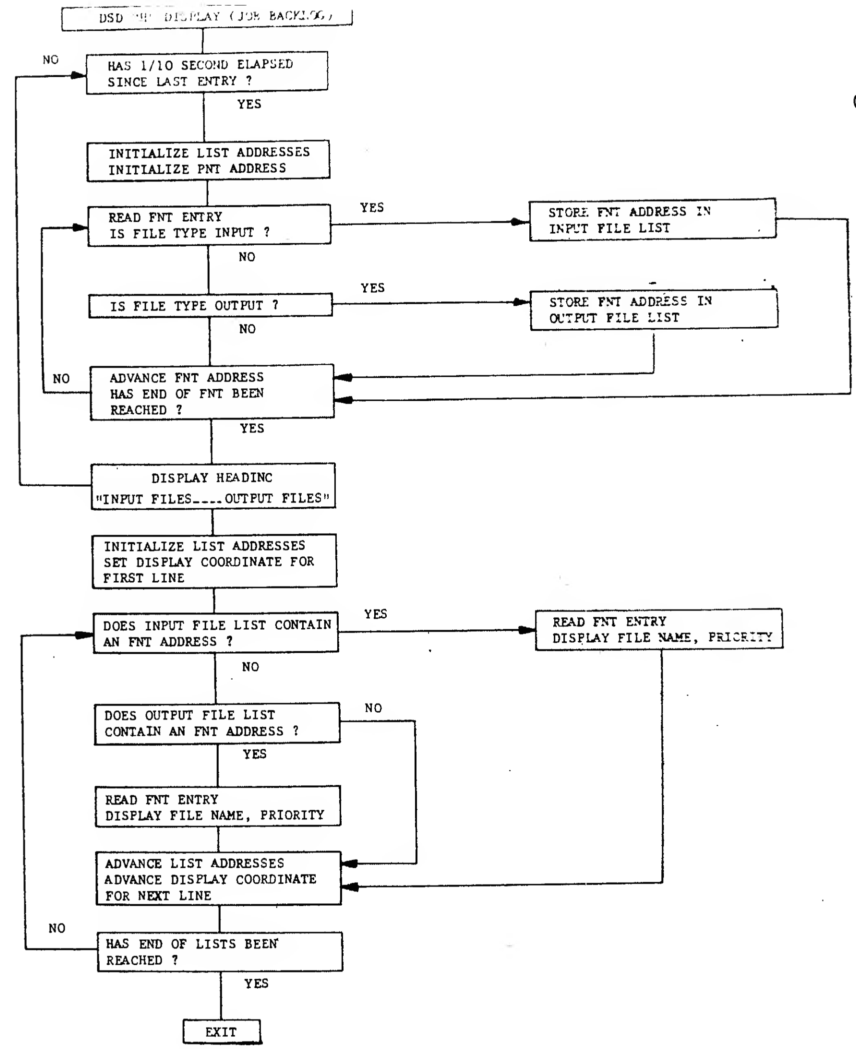



DISPLAY
READ CONTROL POINT STACK
SET STATUS INDICATORS (A-G)
FOR EACH CONTROL POINT IN
THE STACK
READ CONTROL POINT STATUS
IS W FLAG SET ?
YES
SET STATUS INDICATOR FOR
THIS CONTROL POINT TO W
NO
IS X FLAG SET ?
YES
SET STATUS INDICATOR FOR
THIS CONTROL POINT TO X
NO
HAVE ALL CONTROL POINTS
BEEN SCANNED ?
YES
NO
ADVANCE CONTROL POINT
ADDRESS
SET DISPLAY COORDINATE
FOR FIRST CONTROL POINT
DISPLAY CONTROL POINT
NUMBER
READ CONTROL POINT AREA
DISPLAY JOB NAME
DISPLAY PRIORITY
DISPLAY TIME LIMIT
DISPLAY RUNNING TIME
DISPLAY STATUS INDICATOR
READ CONTROL POINT STATUS
DISPLAY PPU ASSIGNMENTS
ADVANCE DISPLAY COORDINATE
FOR NEXT LINE
READ CONTROL POINT STATUS
DISPLAY RA AND FL
READ CONTROL POINT AREA
DISPLAY EQUIPMENT
ASSIGNMENTS
ADVANCE DISPLAY COORDINATE
FOR NEXT LINE
READ CONTROL POINT AREA
DISPLAY LAST DAYFILE MSG
HAVE ALL CONTROL POINTS
BEEN DISPLAYED ?
YES
EXIT
NO
ADVANCE ADDRESS FOR NEXT
CONTROL POINT
ADVANCE DISPLAY COORDINATE
FOR NEXT CONTROL POINT


DSD
NO
HAS 1/10 SECOND ELAPSED
SINCE LAST ENTRY ?
YES
INITIALIZE LIST ADDRESSES
INITIALIZE FNT ADDRESS
READ FNT ENTRY
IS FILE TYPE INPUT ?
YES
STORE FNT ADDRESS IN
INPUT FILE LIST
NO
IS FILE TYPE OUTPUT ?
YES
STORE FNT ADDRESS IN
OUTPUT FILE LIST
NO
NO
ADVANCE FNT ADDRESS
HAS END OF FNT BEEN
REACHED ?
YES
DISPLAY HEADING
"INPUT FILES----OUTPUT FILES"
INITIALIZE LIST ADDRESSES
SET DISPLAY COORDINATE FOR
FIRST LINE
DOES INPUT FILE LIST CONTAIN
AN FNT ADDRESS ?
YES
READ FNT ENTRY
DISPLAY FILE NAME, PRIORITY
NO
DOES OUTPUT FILE LIST
CONTAIN AN FNT ADDRESS ?
NO
YES
READ FNT ENTRY
DISPLAY FILE NAME, PRIORITY
ADVANCE LIST ADDRESSES
ADVANCE DISPLAY COORDINATE
FOR NEXT LINE
NO
HAS END OF LISTS BEEN
REACHED ?
YES
EXIT

CONTROL DATA CORPORATION

Development Division - Applications

# THE JOB DISPLAY, DIS

Chippewa Operating System

10/15/65

# DIS JOB DISPLAY

INTRODUCTION

DIS is the name of the Chippewa Operating System peripheral program that monitors console - keyboard activity for a job assigned to a particular control point. DIS must be loaded in as many PPU units as the number of control points for which it is required. The package is usually located on the system disk in the peripheral library; therefore, its name will appear in the PLD (Peripheral Library Directory). DIS may be brought to a control point in any one of three ways:

1. Typing "A. DIS CR" when DSD (system display) is active.
2. Inserting a DIS control card.
3. A central memory program requesting DIS through a call to MSG.

DIS is concerned with the following functions:

- Job Displays
- Monitoring Keyboard Activity
- Processing Requests for job control debugging for only the job assigned to its control point.

DIS operates during the time the job to which it is attached has the control point. If it is desired to manually release DIS, a drop request is made, which in turn causes a drop PPUs to be issued by the PPU containing DIS. DIS would have to again be loaded for future use by this control point. If an error condition (error byte becomes non-zero in CP area) the program will drop itself; a check of this nature is made on each iteration through the master control loop.

OPERATIONS

CONSOLE DISPLAY

One of the prime functions of DIS is displaying information concerning the status of the job at the control point to which it is attached. To do this, DIS outputs information in the form of display coded characters (see SIPROS DEF Manual) and necessitates issuing X and Y coordinate

values followed by the string of 6-bit display code characters. The screen of each CRT may be considered a grid of points as follows:

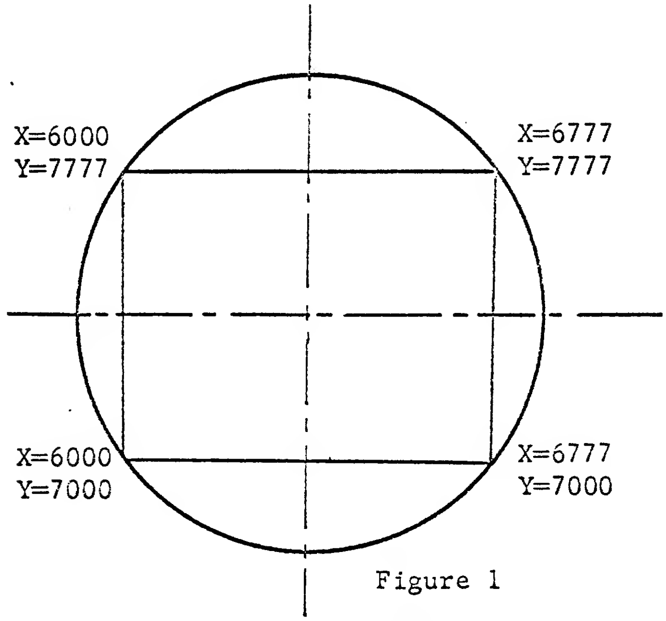



Figure 1

The coordinates (x=6xxx, Y=7xxx) specify the position of the first display coded character follow. Thereafter, the x coordinate is advanced (by one character space) along the increasing x axis but the y coordinate remains constant until another y coordinate (7xxx) is issued. For example, the dayfile display program in DSD uses the area 7200 to 7660 and form 6000 to 6777.

In the dayfile display, the y coordinate is allowed to increase by +1, from 7646 to 7660 on each cycle of the master control loop. As will be seen, this has the effect of "rolling" the display upwards on the screen. The use of the console display is quite simple, involving only the outputing of appropriate x and y coordinates followed by the display coded string of characters.

Besides the formation of display characters and screen positioning, the program also controls the brightness (or intensity) of the image on the screen. The latter increases in proportion to the number of times per second the display coded information is presented to the console. To maintain a stable visable image, the code must be output to the console at least every 1/25 of a second. More repetitions per second will produce a brighter image. A delay loop is commonly employed to control the image output period.

Example of display loop:

```
        LJM   *+3, 10B     .Jump if channel 10B inactive
        DCN        10B     .Disconnect channel 10B
LOOP    FNC   7001B,10B    .Select 32 Char/Line left screen
        ACN        10B     .Activate channel
        LDC   7000B        .A= Y coordinate
        OAN        10B     .Output Y coordinate
        LDC   6337B        .A= X coordinate
        OAC        10B     .Output X coordinate
        LDC         16     .A= No. Words to output
        OAM   Buffer,10B   .Output from buffer
        LDN        01B     .Set A= No. milliseconds delay
        SHN          9     .Convert for LOOP
DELAY   SBN          1     .2 us delay loop
        PJN   Delay
        LJM    Loop        .End of millisecond loop.
```

DISPLAY PLACEMENT

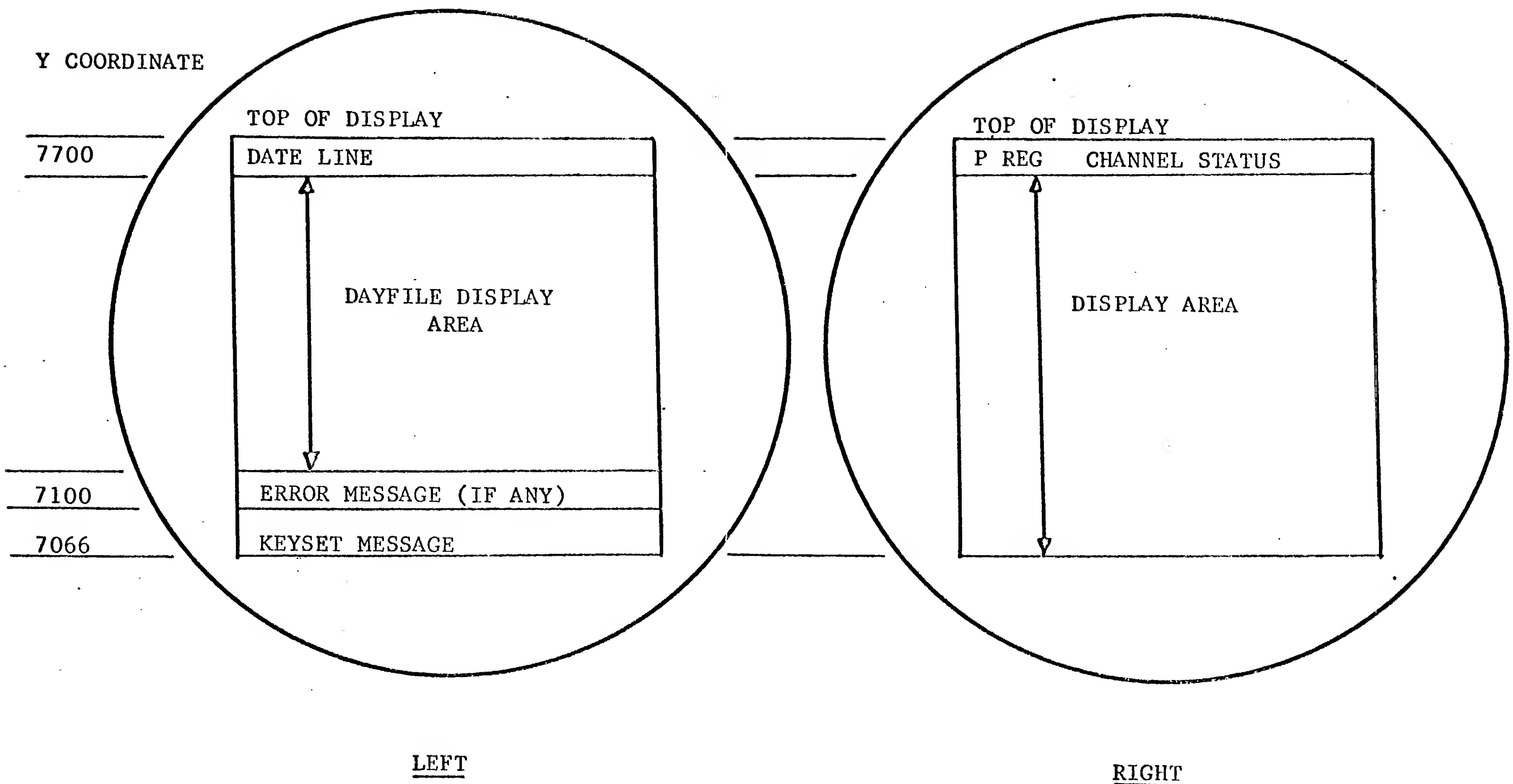



Figure 3

The logic in DIS which controls console output is not basically different from the above example. The overall scheme can be visualized as follows:

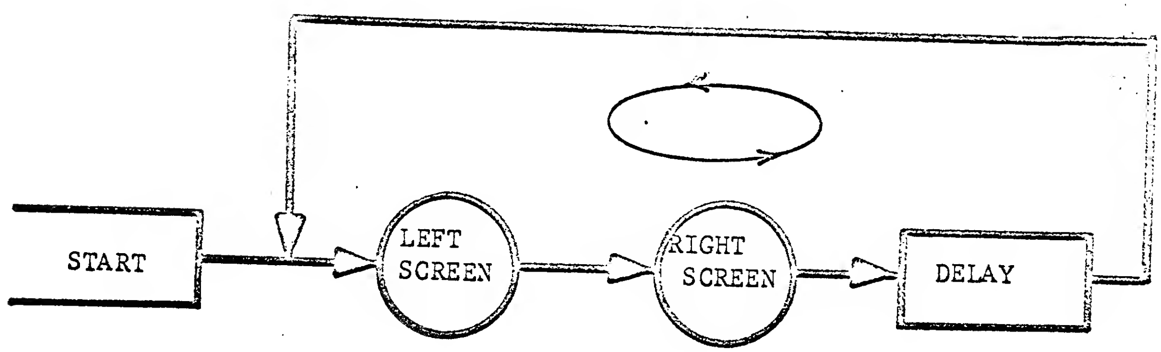



Figure 4

To maintain the delay function, a parameter may be inserted into the ADJUST DISPLAY PERIOD and PRESET INITIAL VALUES ROUTINES. It has the form: LDN nn, where nn is the number of milliseconds (from 00 to $47_8$) in the delay. For maximum brightness, this is preset to nn=00; it can be modified, however. Since other operations (keyboard monitoring) take place in DIS, this constant also effects the total sensitivity of the whole system. For normal operations, it is not necessary to alter this constant. Notice in figure 4 the image is established through continuous trips around the display loop.

KEYBOARD

Along with the display screens at the console unit is a typewrite-like keyboard containing keys for alphabetic, numeric and special characters. These, in display code, are listed on page 46 of the manual on CODES for the 6000 Computer System. Keys are also present for the following special purposes:

| KEY | DISPLAY CODE |
|---|---|
| Carriage Return (CR) | $60_8$ |
| Backspace | $61_8$ |
| Space | $62_8$ |

These keys are used by the KEYBOARD MONITORING routine to control the proper filling of the KB (keyboard) assembly buffer (location 1300/1377 in DIS). A CR is interpreted by DIS as an end of message. The backspace key will erase the last character input. The drop key (code 55) will cause the string of characters of the current message,input to that time, to be cleared; the next character keyed will be treated as the first of a new message.

When it is desired to interrogate the keyboard for keyed information, an input to A is given. If there was a character keyed, the console unit controller will return the character, in display code, in the lower 6 bits of a 12-bit byte (the high order 6 bits are cleared to 00). If no key was activated, however, the controller will return an all zero 12-bit byte.

On each cycle of the Master Control Loop, DIS examines the data at the keyboard. If a valid character is input, the byte is stored away in a contiguous manner in the KB assembly buffer. A cleared byte (0000) will cause no data to be stored. Appropriate action is taken when the CR, backspace or drop key is activated. When a CR is recognized, a flag, called the keyset Ready flag is set to 0001; representing the "on" condition. This flag is examined once during each control cycle. If not set (0000) no action is taken; if set (0001), however, the message in the KB Assembly buffer is interpreted, processed and the flag reset to 0000.

Figure 5 illustrates the position of the keyboard processing in the control sequence of DIS (see figure 4 also):

FUNCTIONAL SEQUENCE OF DIS

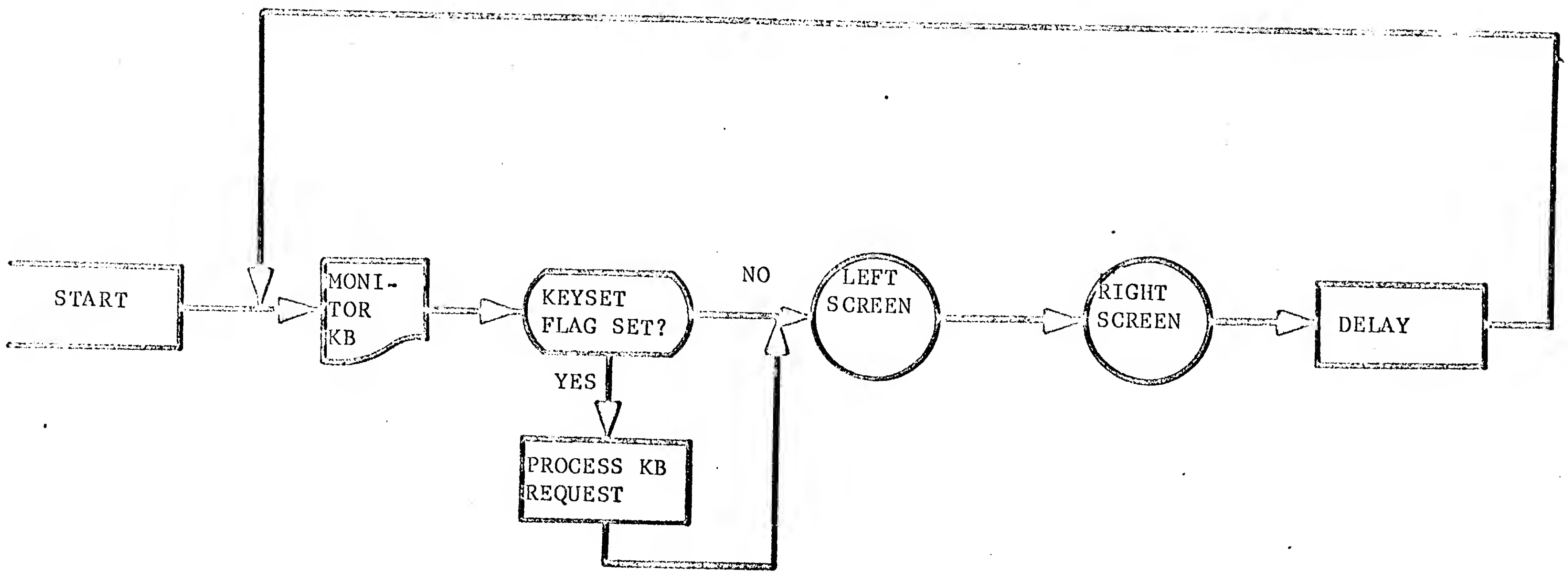



Figure 5

(7)

MASTER CONTROL PROGRAM

Figure 5 combines several of the functions which make up the MCP (MASTER CONTROL PROGRAM):

- Keyboard Monitoring
- Request Processing initiation
- Left Screen Display
- Right Screen Display
- Display Period Adjustment

Only a few more functions need to be added to produce the operations performed by the MCP. Regardless of the display mode selected (A, B, ..., G) there are permanent displays on both the left and right screens. On the left appears the time and date line stored in Central Memory locations 30/37; this is displayed at Y coordinate 7700. At Y coordinates 7100 and 7066 are the error message (if any) and the contents of the KB Assembly Buffer (at that point in time), respectively. The current Central Processor P register and the status of all 12 data channels are displayed at Y coordinate 7700 on the right screen. In the functional diagram (figure 5) these displays may be thought of as belonging to the left and right screen displays.

One function remains: breakpoint monitoring. The user is given the opportunity to request (via the keyboard) a breakpoint debugging action. This routine reads the CP P register and if the contents match the requested breakpoint address (in P 52/53) the Central Processor is dropped from the control point and the word at the breakpoint (saved in P40/144 during breakpoint initiation) is restored in CM. This function follows the display period adjustment routine. All functions of the MCP are illustrated in detail in MCP flow chart (see appendix A).

DISPLAY PROGRAMS

DIS is able to provide four different types (modes) of displays. The operations indicated in the two circles in Figure 5 refer to the particular display and its associated display program selected for that screen for discussion purposes. The permanent displays were considered here as well. Initially, DIS will put up the dayfile display on the left screen and the job status infor on the right. These can be altered, if desired, by keying the following request.

LR. "CR"

The mode code placed in the L position will bring the display for that code to the left screen and one placed in R will specify the right screen display. The address for the left and right screen programs are stored in locations 70 and 71, respectively.

DISPLAY MODES

| CODE | DISPLAY INFORMATION |
|---|---|
| A | Day File |
| B | Job Status |
| C, D, E | Program Storage |
| F, G | Data Storage |

Day File Display - MODE A

This program displays the contents of the DFB (Day File Buffer) between Y coordinates 7660 (top) to 7200 (bottom). This coordinate is stored in location 64. The most current being displayed message will always appear at the bottom of the display. Up to 3/10 messages can appear between these limits. After each message is displayed, the Y coordinate is decremented by $12_8$.

A pointer in location 65 is maintained to indicate the address of the DFB message to be displayed at the top of the display. Initially this pointer equals the "OUT" address of the DFB status word (CM location 0003); this is

advanced as the number messages between the current address (at 65) and the "INPUT" address exceeds $31_{10}$. When 31 or fewer DFB messages are between (65) and "INPUT", the display stays constant with new messages being displayed at the bottom of the display as they appear in the DFB.

If, however, the number of messages in the DFB exceeds the maximum number that can be processed at one time, the display will not remain fixed. Rather, it will "roll up" on the face of the screen. This gives the impression that when the message now at the head of the display reaches the top it is rolled off the screen and a new (and more current) message enters the display at the bottom. This will continue until the number of DFB messages between the address stored in 65 and the INPUT DFB status indicator becomes less than 31. This rolling is accomplished by allowing the beginning Y coordinate to vary from 7646 to 7660 (in increments of 1) on each cycle of the DIS master control loop. Therefore, $12_8$ iterations will be necessary to roll off a message. As each message is rolled off the screen, the DFB pointer in 65 is advanced to the address of the next message in the DFB. All display references are made relative to this address. Eg: if 65 contained 2334 then the first (top most) message would be picked up from the CM location 002334. Then the next 30 messages (not the next 30 cells) to be displayed will be picked up. When the message beginning at CM 002334 is rolled off the display, the address in 65 will be set to point to the first word of the next DFB message. If the message at CM 002334 is three cells long, 65 will be set equal to 002337. Whenever a message is rolled off, the Y coordinate is reset equal to 7646. See the flow chart in Appendix A, A-39 for a detailed description of the process.

<u>Job Status Display</u> - <u>MODE B</u>

This display exhibits the control point status. (W, X, A, .....G, -), last dayfile message the next control state-

ment to be processed and the exchange jump package. This information is gathered from the control point area.

Storage Displays - MODES C, D, E, F, G
Modes C, D, E are primarily used to display program text residing within Central Memory. These form display octal digits in the form of 4 groups of 5 digits each.

Modes F, G, in contrast, display octal digits in 5 groups of 4 digits. These coorespond to the 12-bit PPU words and hence modes F, G are used for data storage.

Besides the above differences, the five modes all share these characteristics. All displays have four fields. A field is the display of the eight words XXXXX0 - XXXXX7. The particular field specification is given by the typed statement:

Xn, m. "CR"

where X = C, D, E, F, G

n = 0 Field 0 begins with m
1 Field 1 begins with m
2 Field 2 begins with m
3 Field 3 begins with m
4 Four consecutive field beginning with m.

m = Central memory relative (to RA) address. This should be of the form XXXXX0. In any case, the low order digits is made 0 if any other digit is specified.

Eg: 1. C2,330. "CR" would set field 2 beginning address equal to 000330 and display 000330 - 000337.

Eg: 2. E3,351. "CR" would set field 3 equal to 000350 and display 000350 - 000357.

All displays give the relative CM address to the left of each entry of the display.

REQUEST PROCESSING

On each iteration through the Master Control Loop the Keyset Ready flag is examined. If the flag has the value

0000 (i.e., not set) the remaining portion of the loop is traversed. If, however, the flag is set, control is given to the INTERPRET KEYBOARD MESSAGE routine. This routine scans the information in the KB buffer and gives control to the proper routine to process the keyed request.

The requests may be classified into the following groups:

1. Display mode selection and mode field specification
2. Central Memory modification
3. Exchange Jump Package and Control Point modification
4. Job control
5. Debugging Aids

The KB interpretive routine first checks for a special format (see list of possible requests following this discussion) and if request is of this gives control to the routine specified (see flow charts A-17 to A-36). If it is not one of these, the statement is examined for a display entry (mode or field change); if it is of this type, control goes to proper display processor. If the statement still can not be identified, it is treated as a possible PP call and the RPL and PLD are reached. If a match is made, a request is set up in 10/14 and a return is made to PP Resident to inform EXEC that there is a request to process. If the request is not a PP call, it is considered to be an error and the message "FORMAT ERROR" is desplayed on the left screen. The Keyset Ready flag is then cleared to 0000 and control is returned to the Master Control Program. For a complete description of individual request processing, consult the DIS flow charts in Appendix A.

**DIS REQUESTS**

The following commands to DIS refer to the control point to which it is attached. Some of the entries cause the job to be switched away from the CPU (e.g. when the job's exchange package has to be changed). Execution can be resumed using RCP or BKP, Numbers are in octal.

| | |
|---|---|
| . ENP, 12345. | Set P = 12345. (Next instruction address in exchange package). |
| . ENA3, 665000. | Set A3=665000 in exchange package. |
| . ENB2, 44. | Set B2=44 in exchange package. |
| . ENX5, 2223 4000-0000 0000 0200. (Spacing unimportant) | Set X5=22234000000000000200 in exchange package. |
| . ENEM, 7. | Set Exit Mode = 7 in exchange package. |
| . ENFL, 10000. | Set FL=100000 en exchange package. (Storage moved if necessary). |
| . ENTL, 200. | Set CPU Time Limit = $200_8$ seconds. |
| . ENPR, 5. | Set job Priority = 5. |
| . DCP | Drop central processor and display exchange package (in display B). Using DIS, the exchange package is displayed in any case if the job does not have status A, B, etc. |
| . RCP. | Request central processor. This puts the job in W status, and it will take the CPU if its priority is sufficient. The register settings of the exchange package will be used. |
| . BKP, 44300. | Breakpoint to address 44300 in the program. CPU execution begins at the current value of P and stops when P = 44300. DIS effects this by clearing 44300 to stop the program at that point, and restores the original word when the stop occurs. |

| | |
|---|---|
| . RNS. | Read next control statement and obey it. (During use of DIS the normal advance of control statements is inhibited). |
| . RSS. | Read next control statement and begin execution. This is like RNS, except that a central program is only brought to central memory, and not executed. |
| . ENS.xxxxxxxxxxxxxx. | This command allows the entry of any control statement as if it had been entered on a control card. The statement can then be processed using RNS or RSS. |
| . GO. | This command restarts a program which has paused. |
| . ONSW3. | Set sense switch 3 for the job. |
| . OFFSW4. | Switch off sense switch 4 for the job. |
| . HOLD. | This entry causes DIS to relinquish its display console, but the job is held at its present status. A console must be reassigned to continue use of DIS. |
| . DROP. | This causes DIS to be dropped and normal execution of the job is continued. It does not mean 'Drop the job.' |
| . DMP (200, 300). | Dump storage from 200 to 277 in the output file. |
| . DMP (400). | Dump storage from the job's reference address to 377. |
| . DMP. | Dump exchange package to output file. |

(DMP formats are the same as if used on control cards).

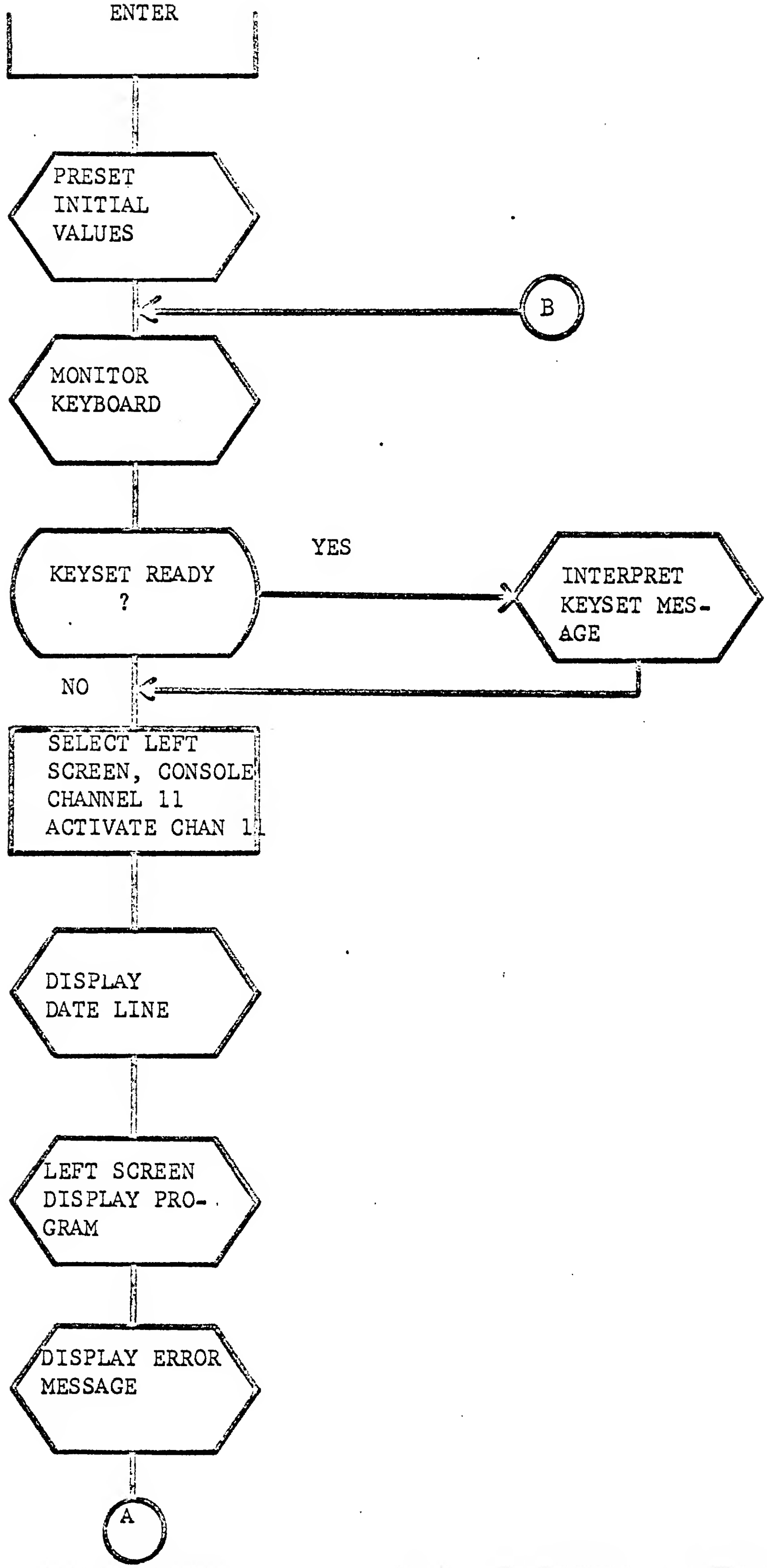
# APPENDIX A - DIS FLOW CHARTS

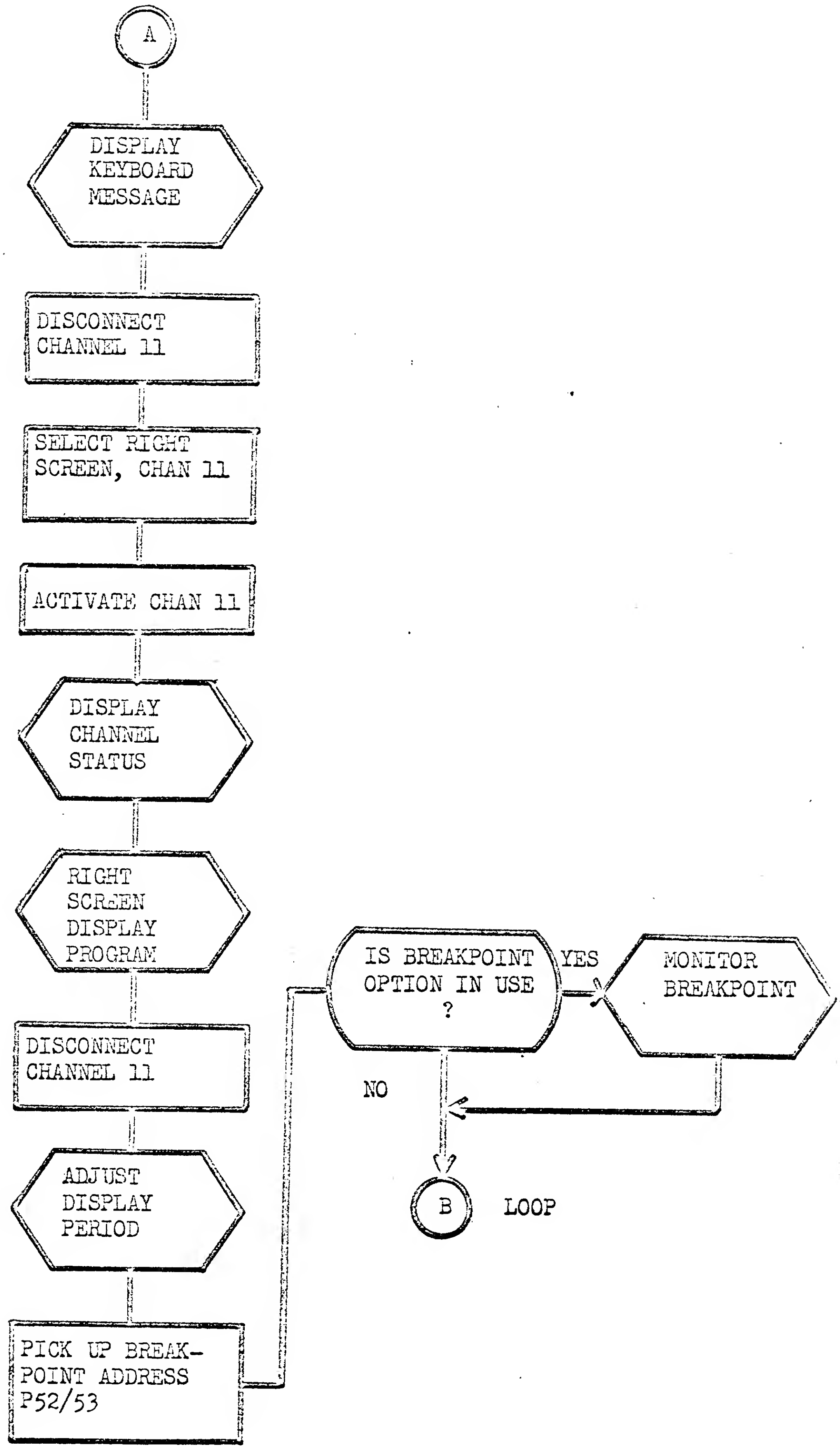
## INDEX

INDEX

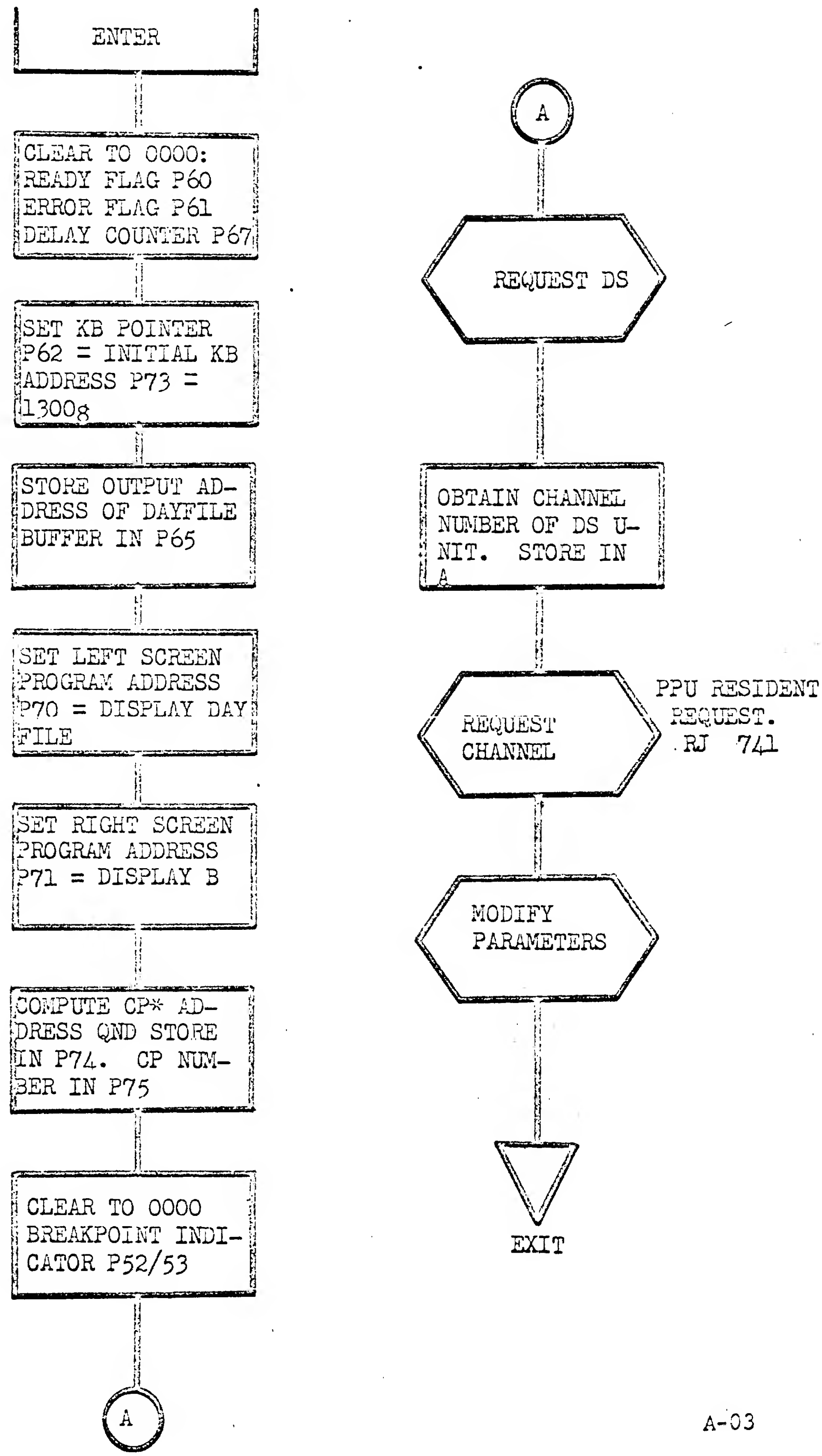
| ODIS | TEMPORARY STORAGE ALLOCATION |
|---|---|
| 40/44 | BREAKPOINT WORD |
| 50 | REFERENCE ADDRESS |
| 51 | FIELD LENGTH |
| 52/53 | BREAKPOINT ADDRESS |
| 60 | KEYBOARD READY FLAG |
| 61 | KEYBOARD ERROR FLAG |
| 62 | KEYBOARD ADDRESS |
| 63 | EQUIPMENT ADDRESS |
| 64 | DAYFILE DISPLAY COORDINATE |
| 65 | DAYFILE DISPLAY ADDRESS |
| 66 | DISPLAY CYCLE COUNTER |
| 67 | DELAY COUNT |
| 70 | LEFT SCREEN PROGRAM |
| 71 | RIGHT SCREEN PROGRAM |
| 73 | KEYSET INITIAL ADDRESS |
| 74 | CONTROL POINT ADDRESS |
| 75 | INPUT REGISTER |
| 76 | OUTPUT REGISTER |
| 77 | MESSAGE BUFFER |

DIS - MAIN CONTROL PROGRAM



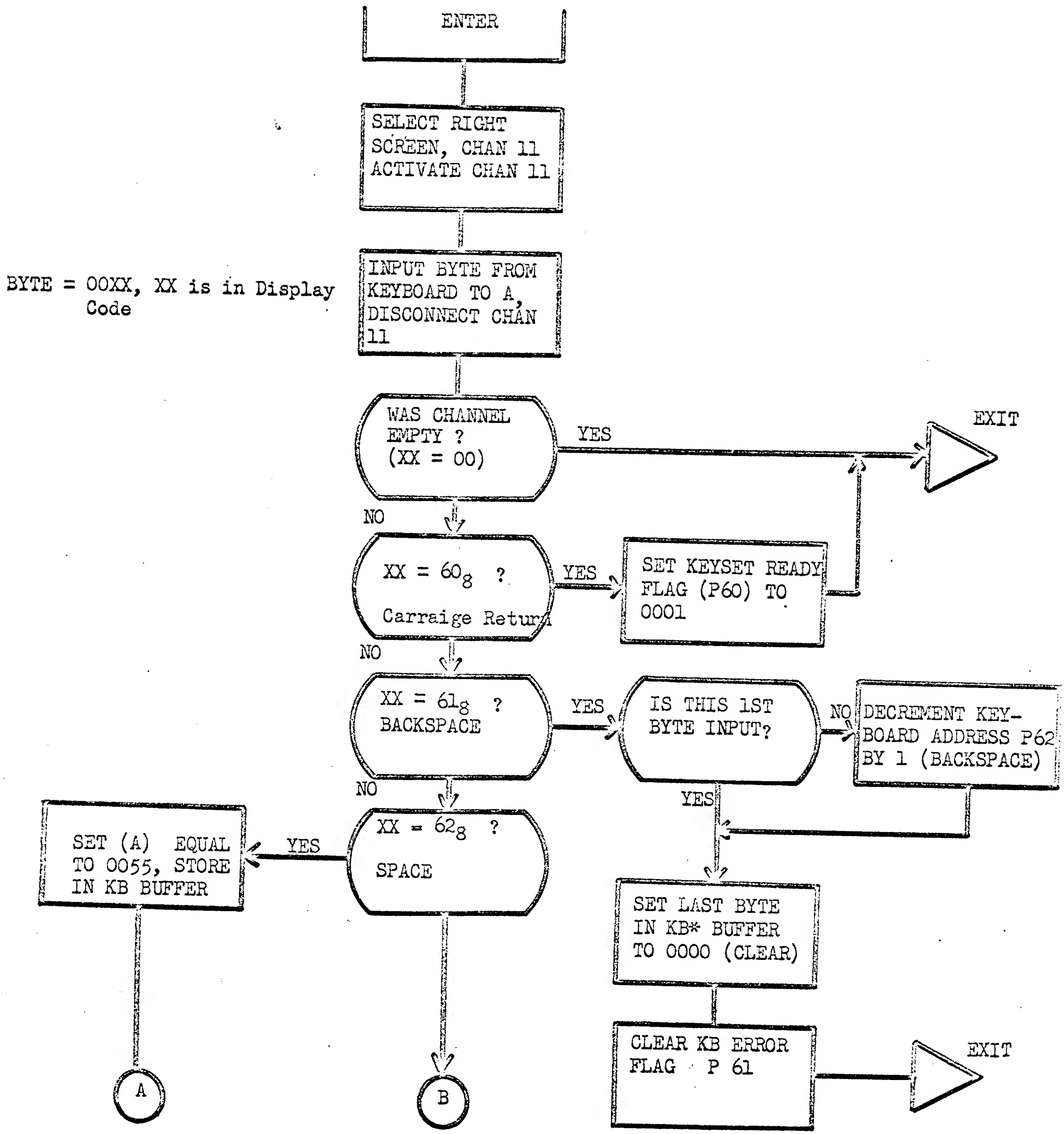

A
DISPLAY KEYBOARD MESSAGE
DISCONNECT CHANNEL 11
SELECT RIGHT SCREEN, CHAN 11
ACTIVATE CHAN 11
DISPLAY CHANNEL STATUS
RIGHT SCREEN DISPLAY PROGRAM
DISCONNECT CHANNEL 11
ADJUST DISPLAY PERIOD
PICK UP BREAK-POINT ADDRESS P52/53
IS BREAKPOINT OPTION IN USE ?
YES
MONITOR BREAKPOINT
NO
B
LOOP

DIS - PRESET INITIAL VALUES



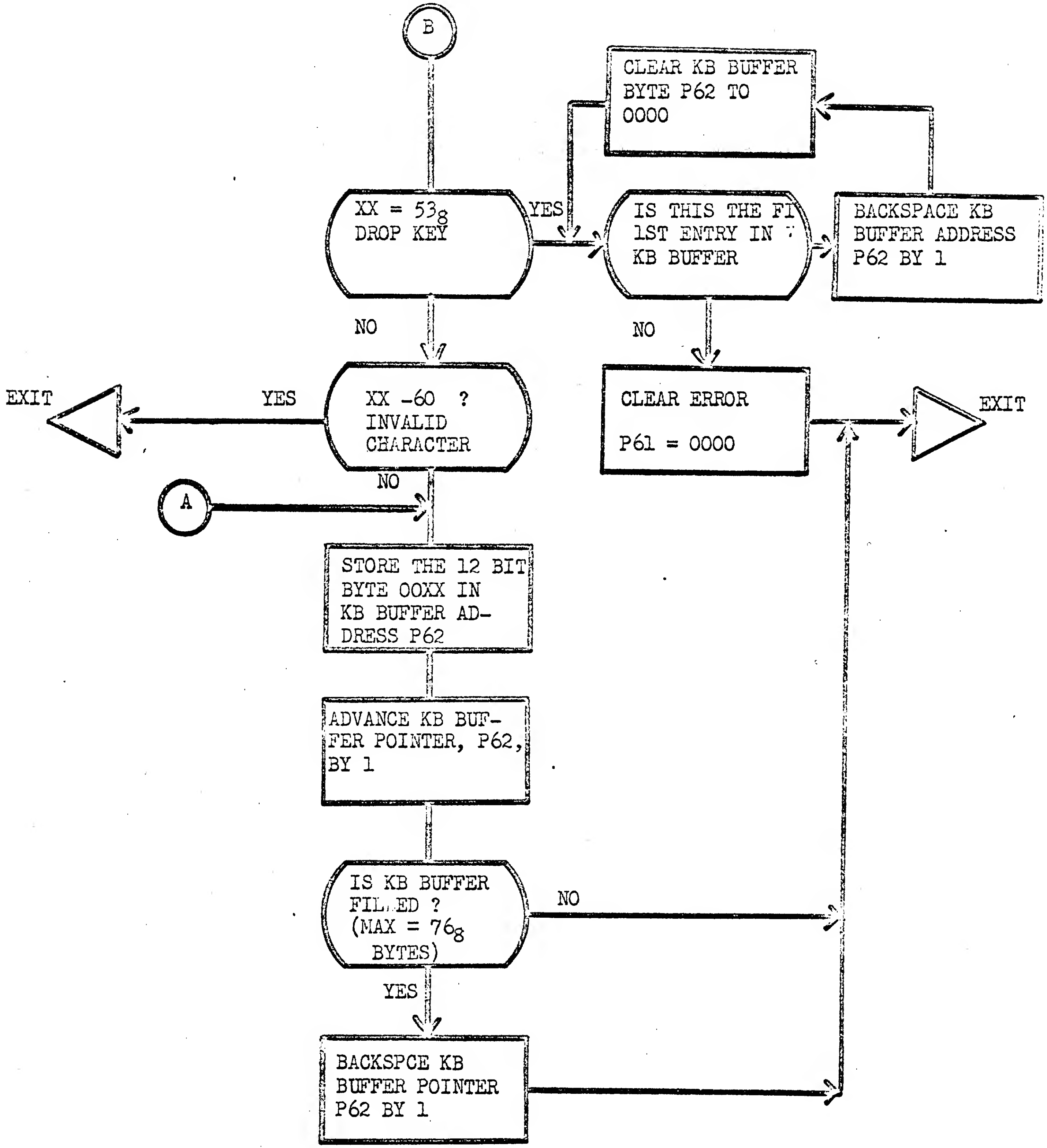

* CP Means Control Point

DIS - MONITOR KEYBOARD

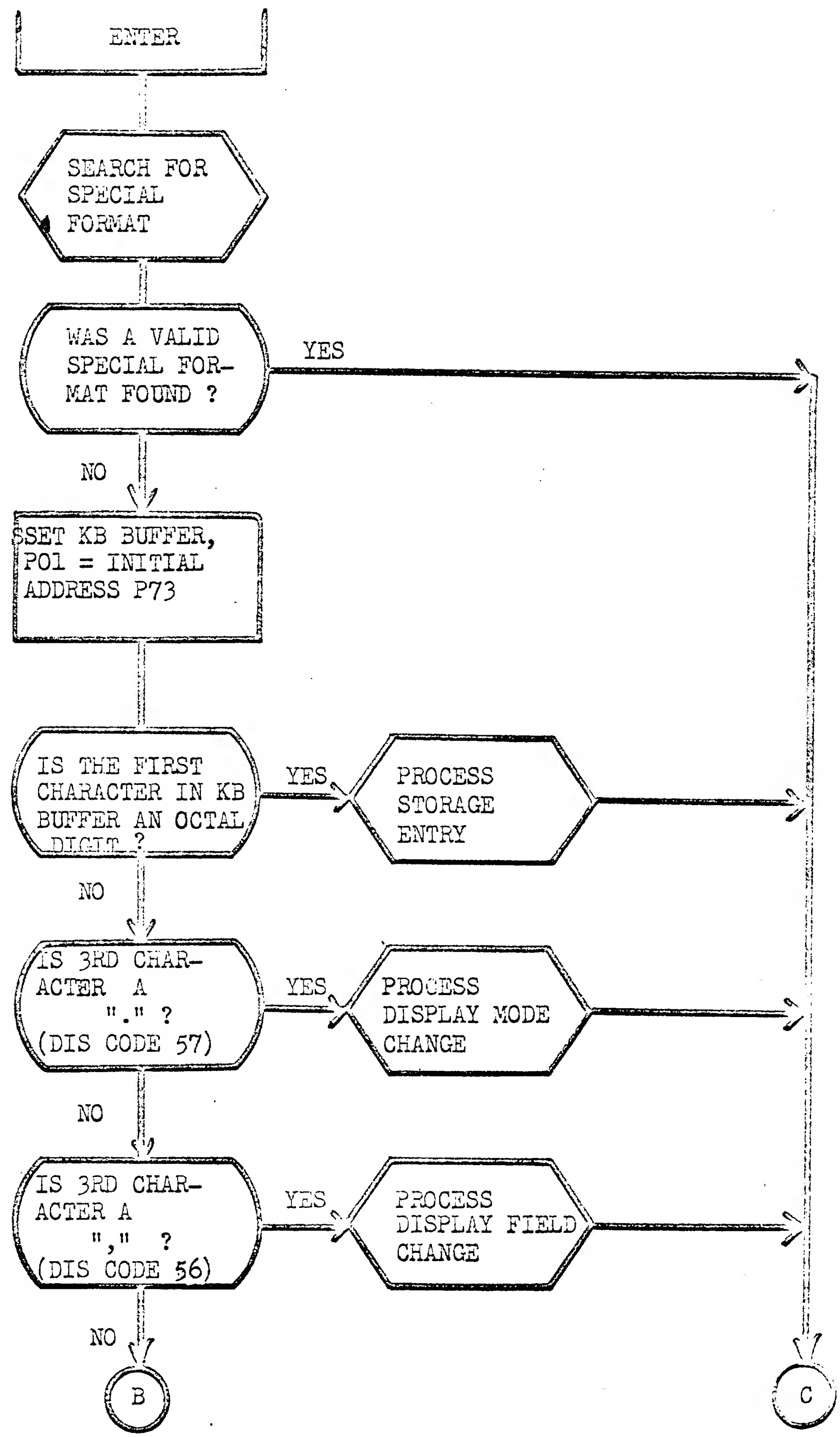



* - KB Means Key Board

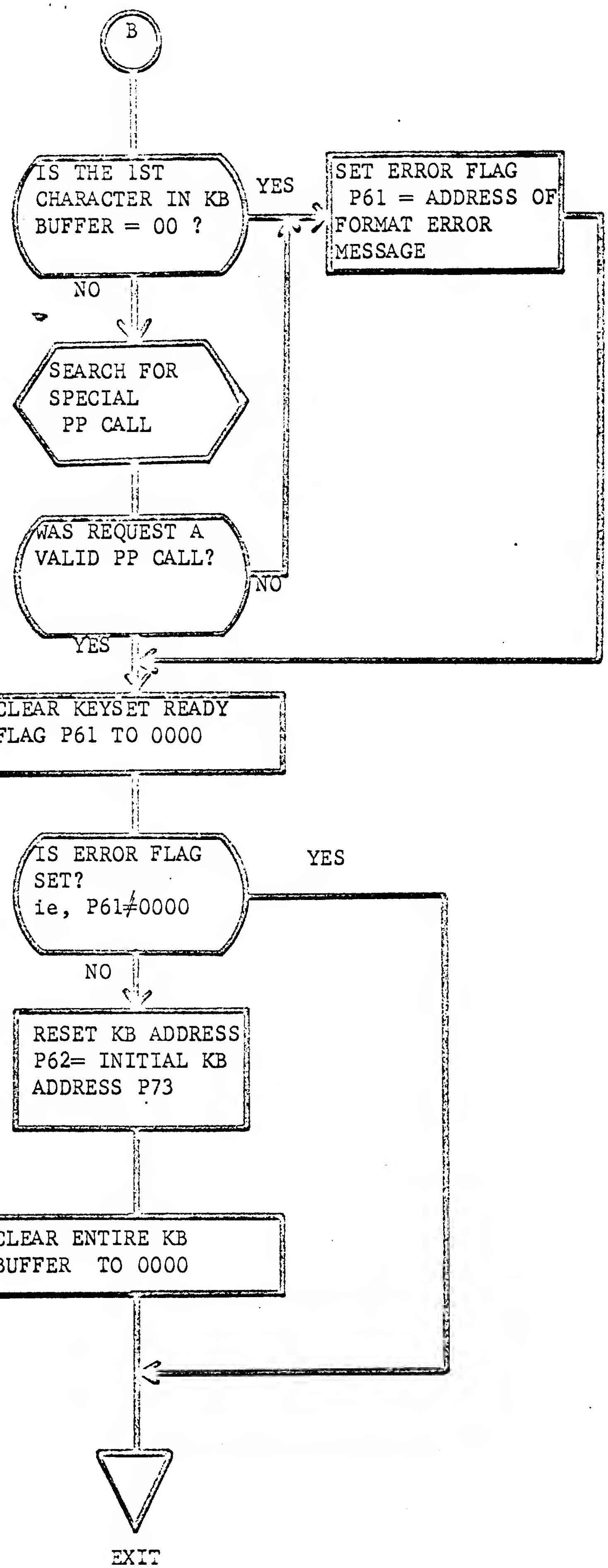
DIS - MONITOR KEYBOARD (CONTINUED)



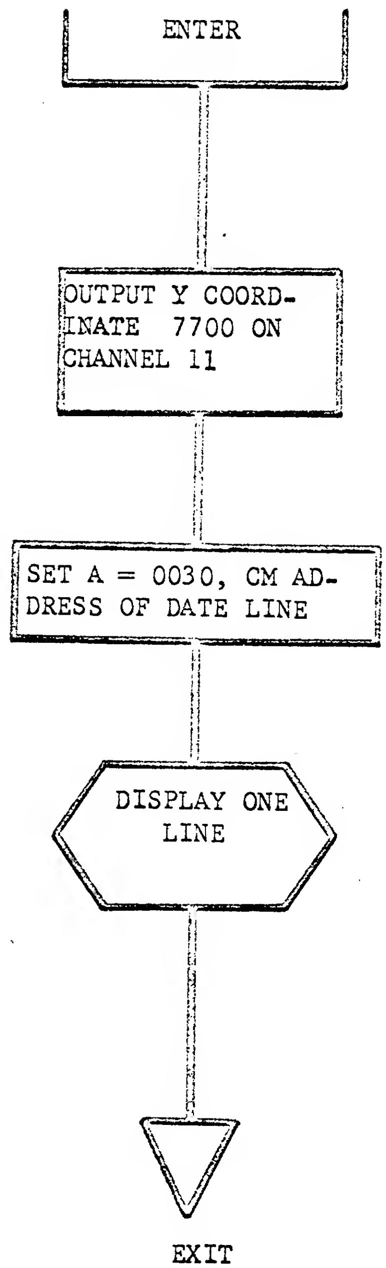
DIS - INTERPRET KEYSET MESSAGE



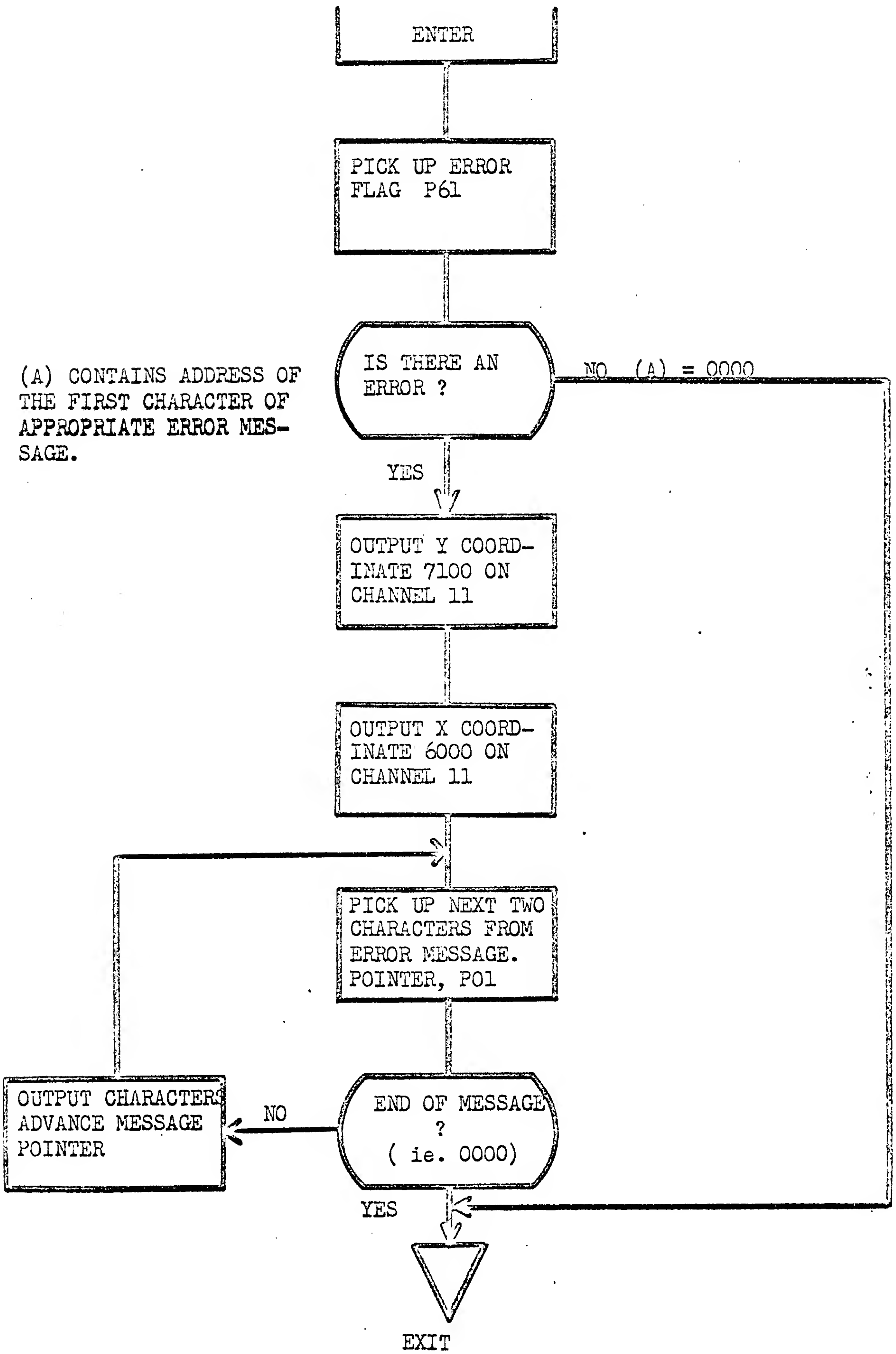
DIS - INTERPRET KEYSET MESSAGE (CONTINUED)



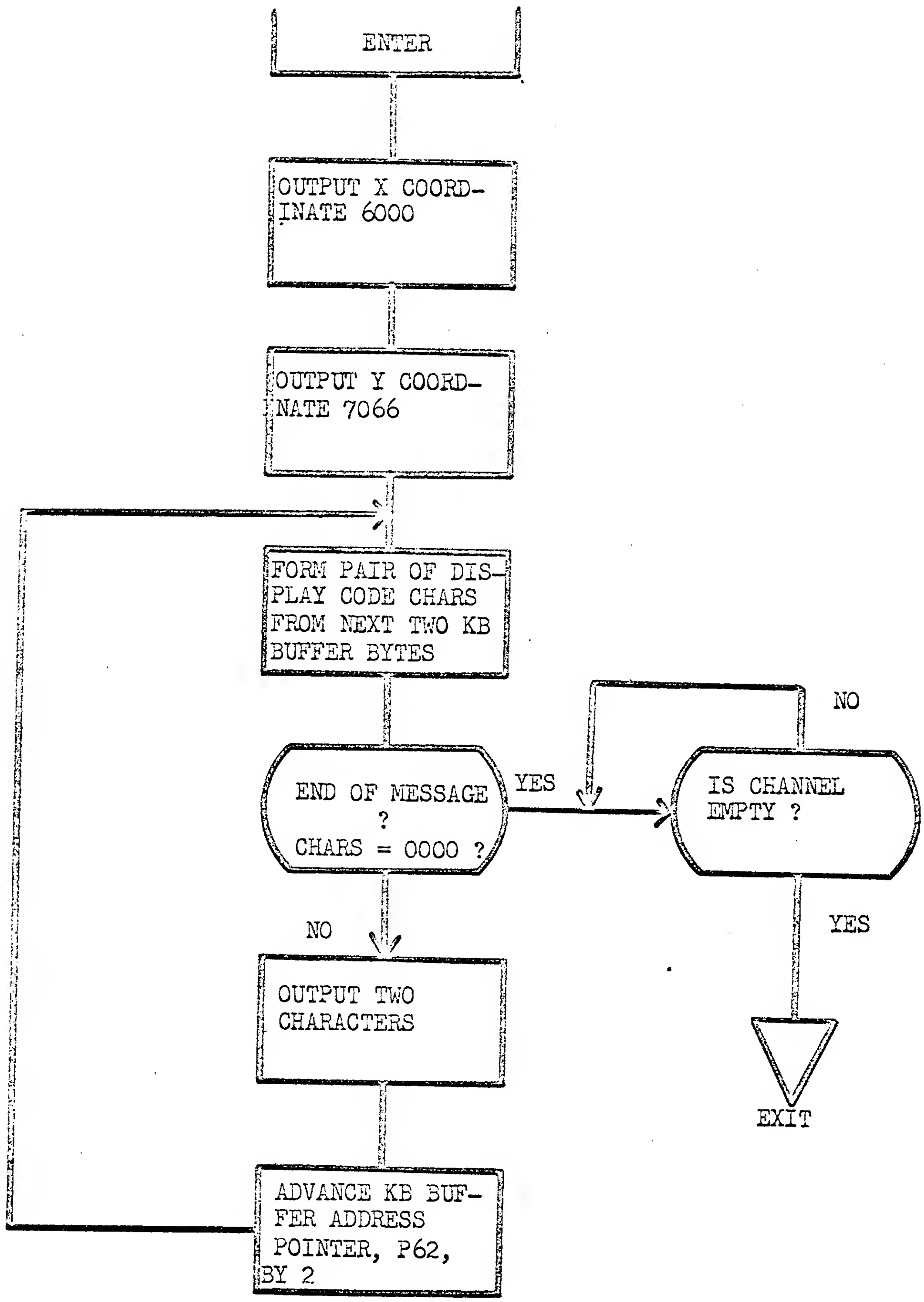
# DIS - DISPLAY DATE LINE



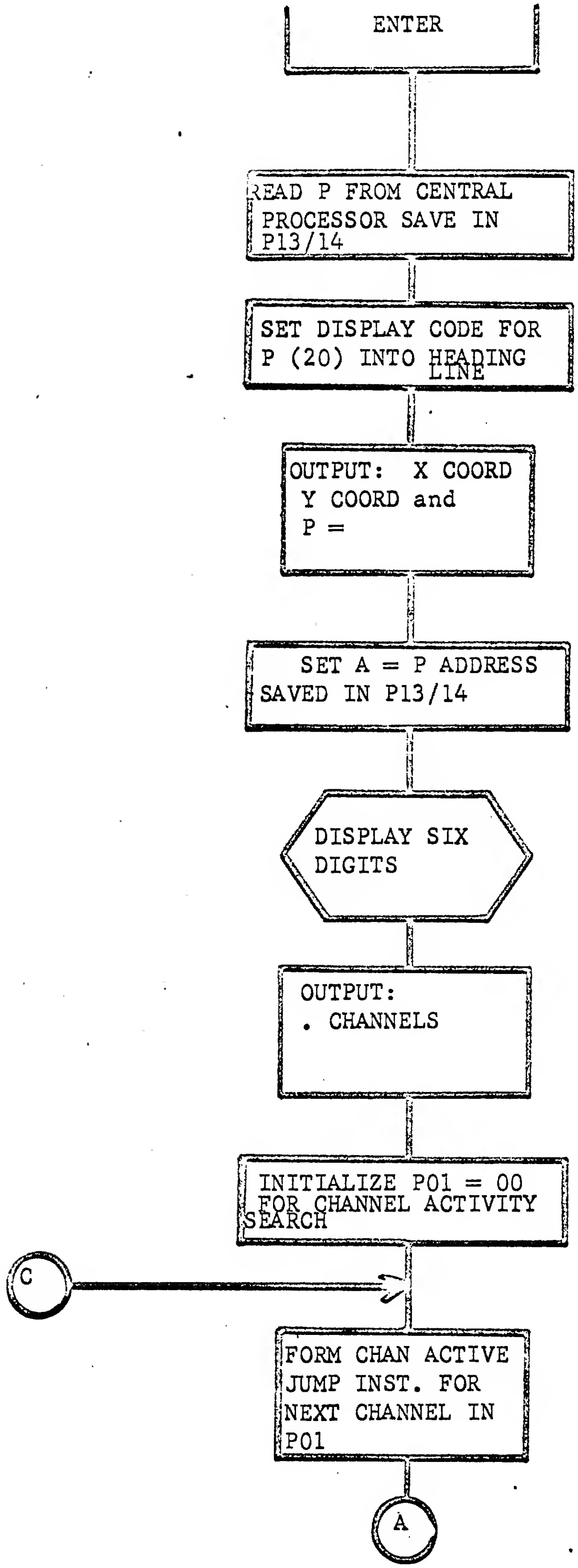
## DIS - DISPLAY ERROR MESSAGE



# DIS - DISPLAY KEYSET MESSAGE

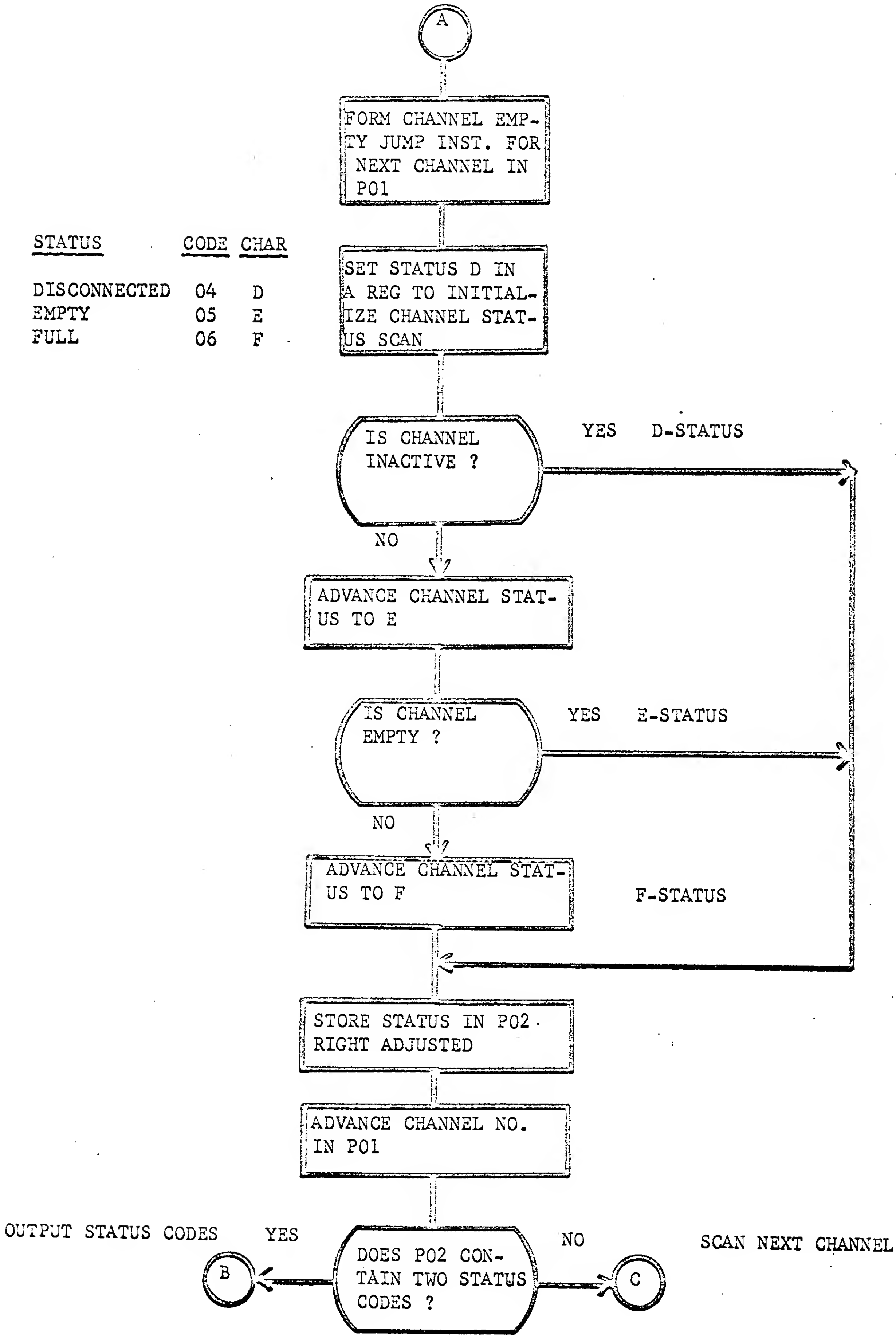


DIS - DISPLAY CHANNEL STATUS

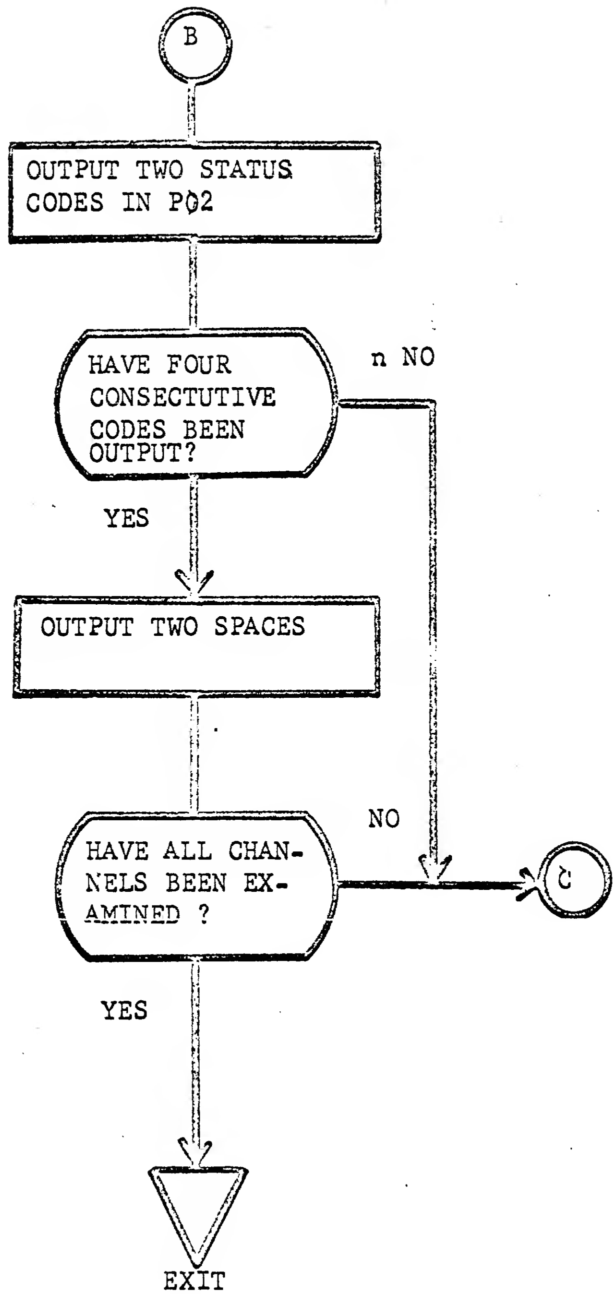


## DIS - DISPLAY CHANNEL STATUS (CONTINUED)

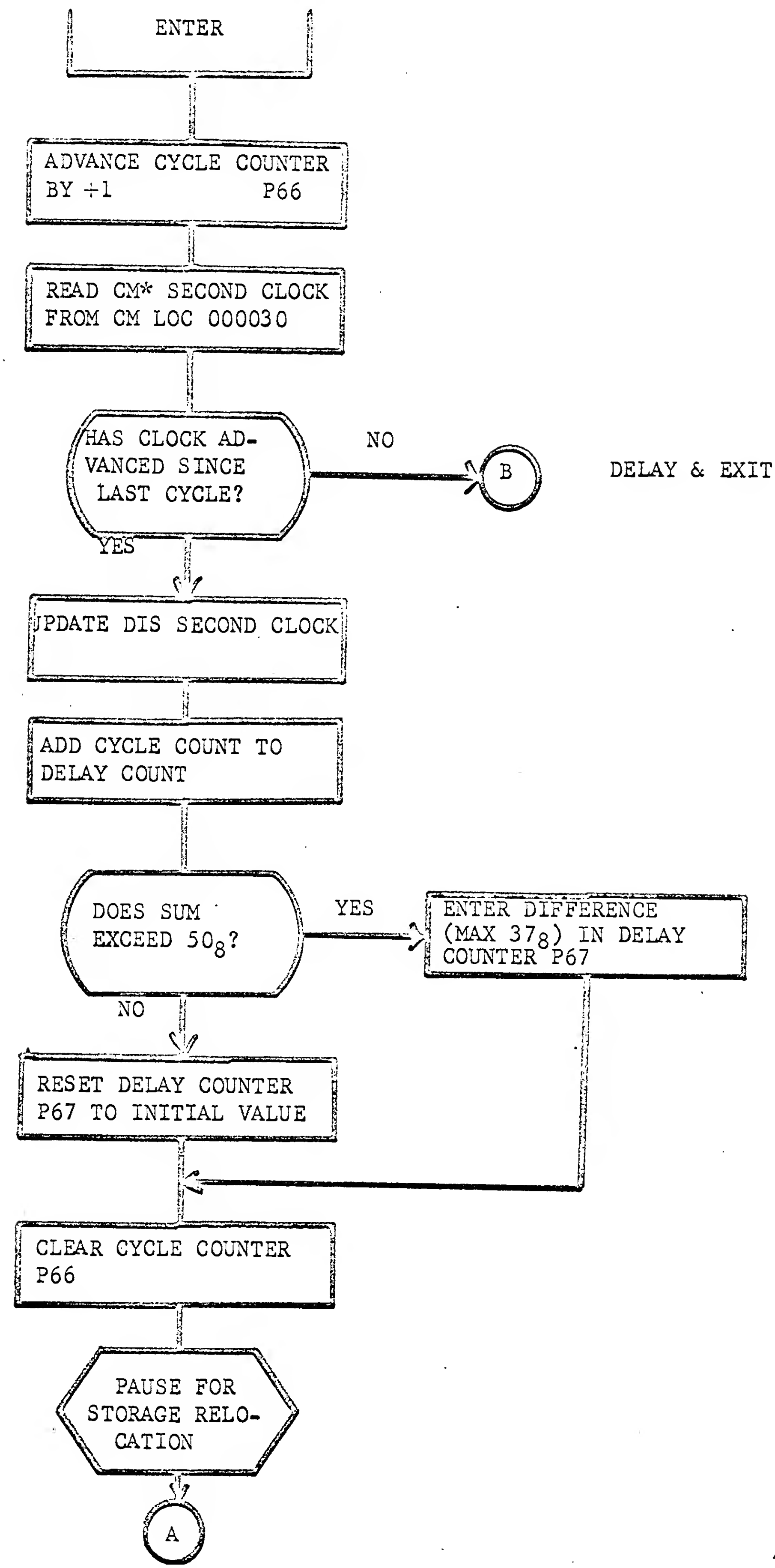
| STATUS | CODE | CHAR |
|---|---|---|
| DISCONNECTED | 04 | D |
| EMPTY | 05 | E |
| FULL | 06 | F |



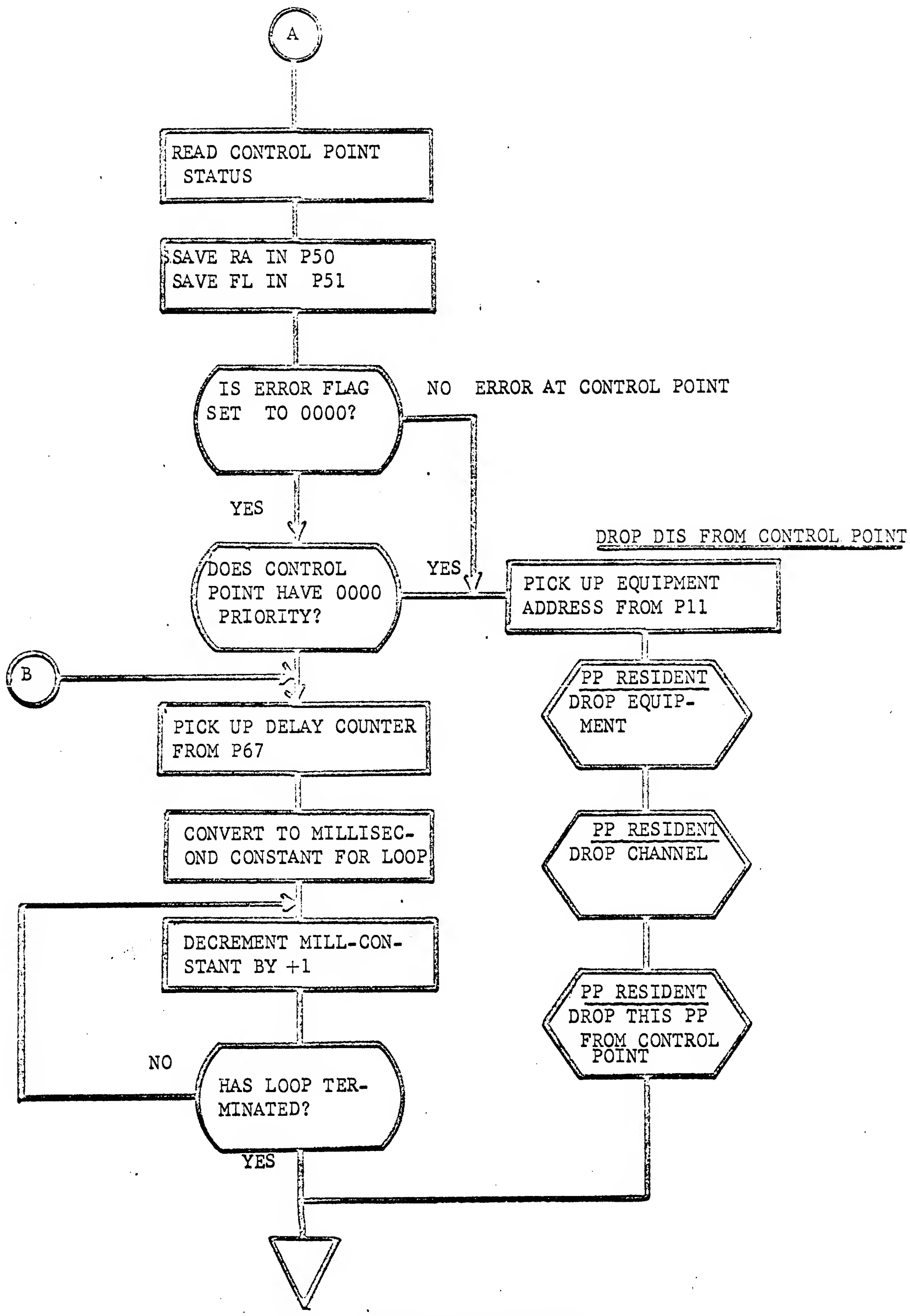
DIS - DISPLAY CHANNEL STATUS (CONTINUED)



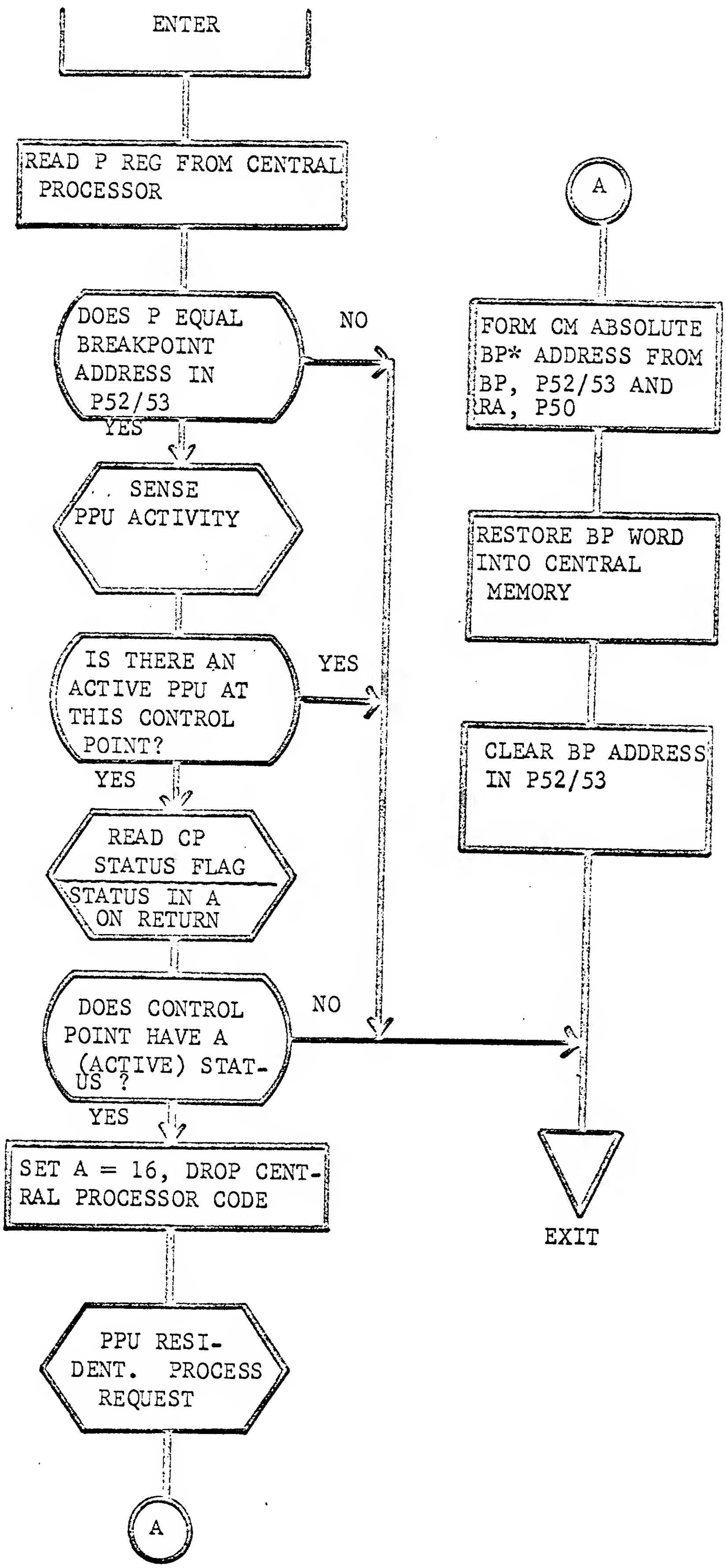
DIS - ADJUST DISPLAY PERIOD



DIS - ADJUST DISPLAY PERIOD (CONTINUED)



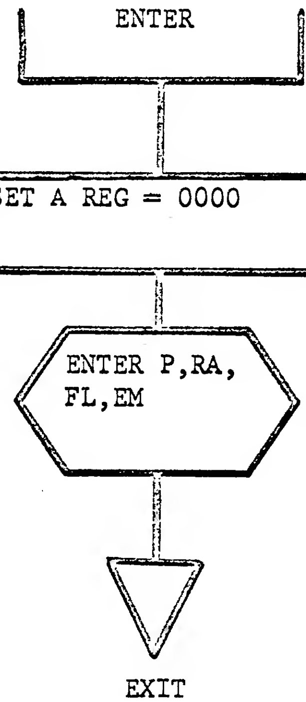
DIS - MONITOR BRAEKPOINT ADDRESS



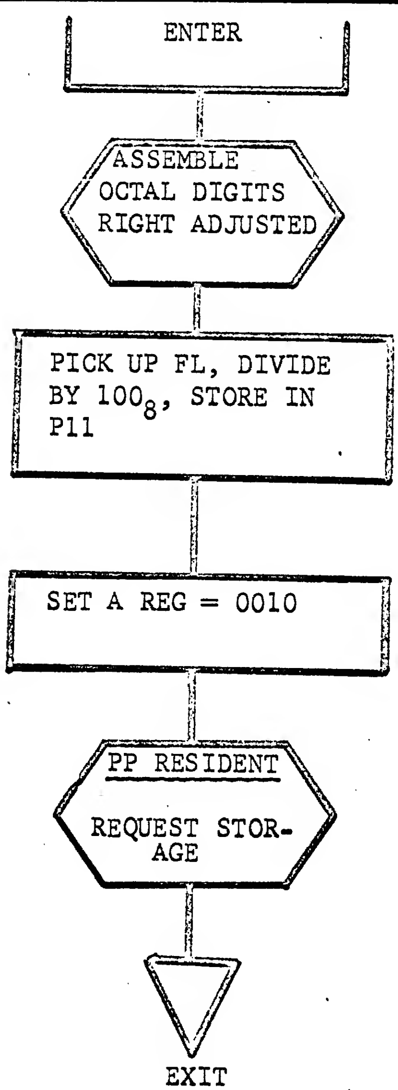

| STATUS | CODE |
|---|---|
| WAITING | W |
| RECALL | X |
| ACTIVE | A |
| IN STACK | B to G |

* BP Means BreakPoint

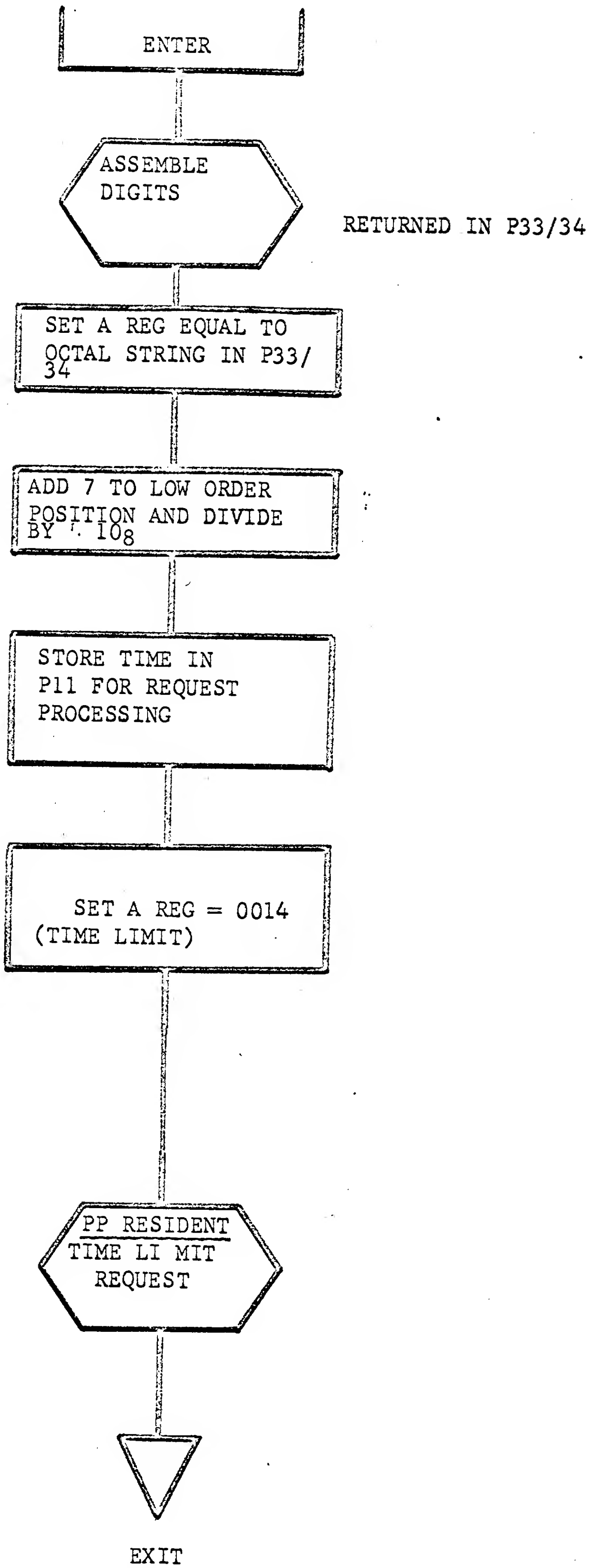
DIS - ENP REQUEST PROCESSOR



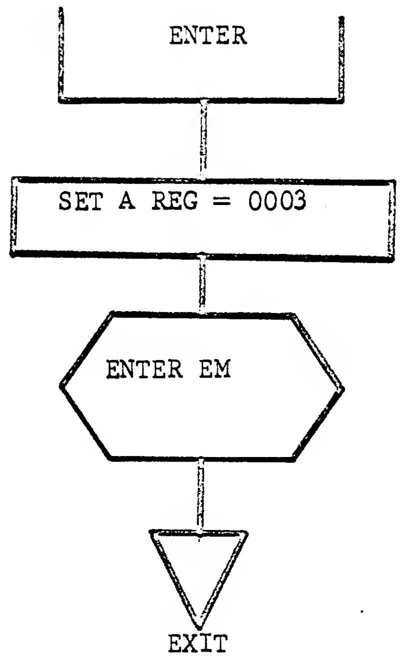

DIS - ENFL REQUEST PROCESSOR



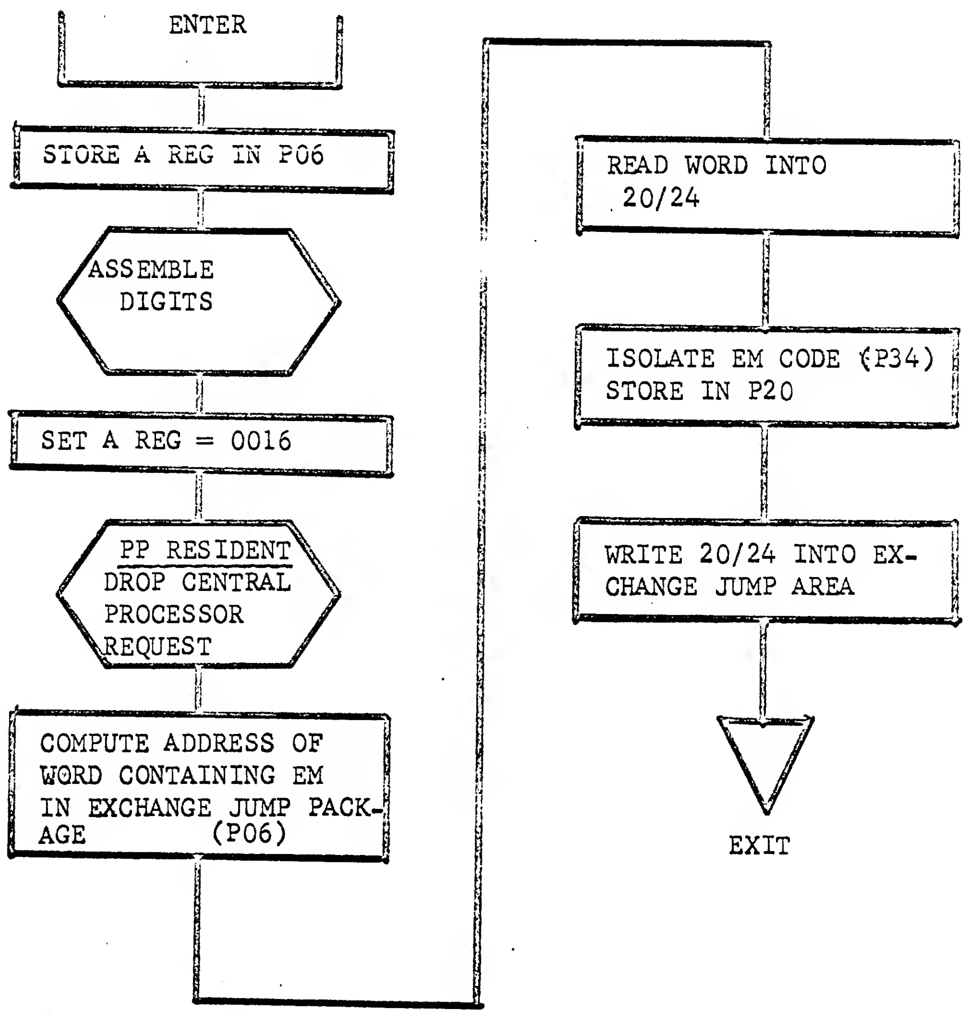
DIS - ENTL REQUEST PROCESSOR



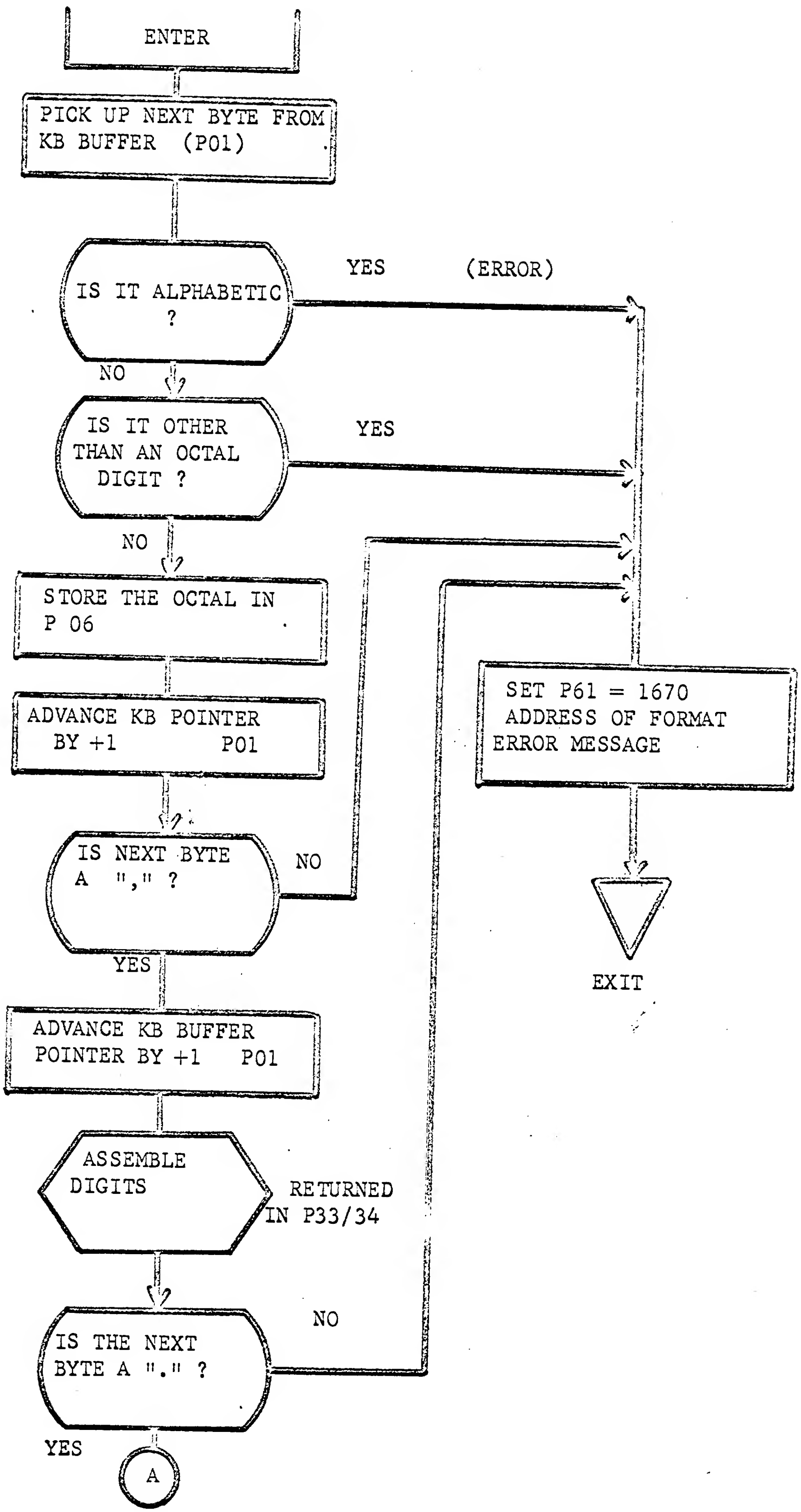
DIS - ENEM REQUEST PROCESSOR



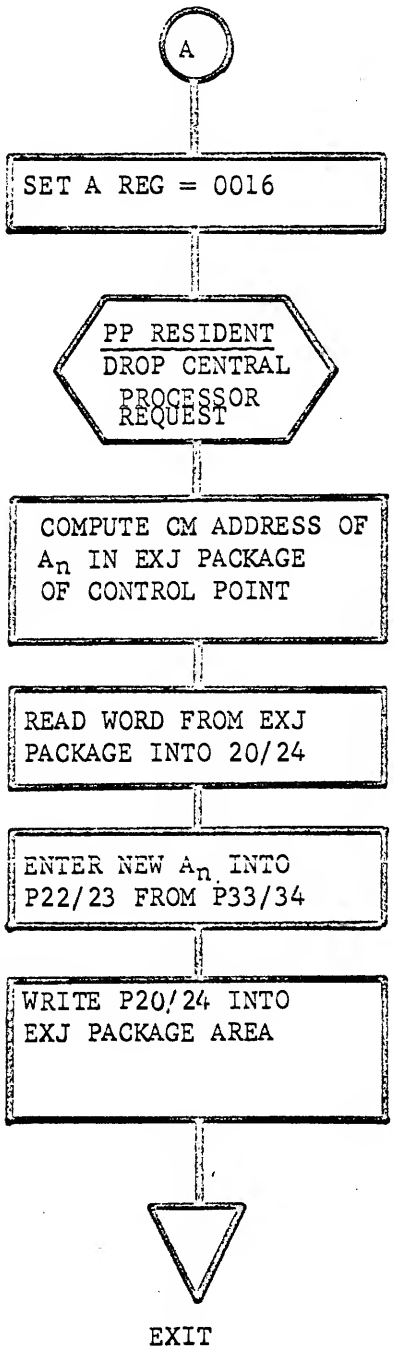

DIS - ENTER EM



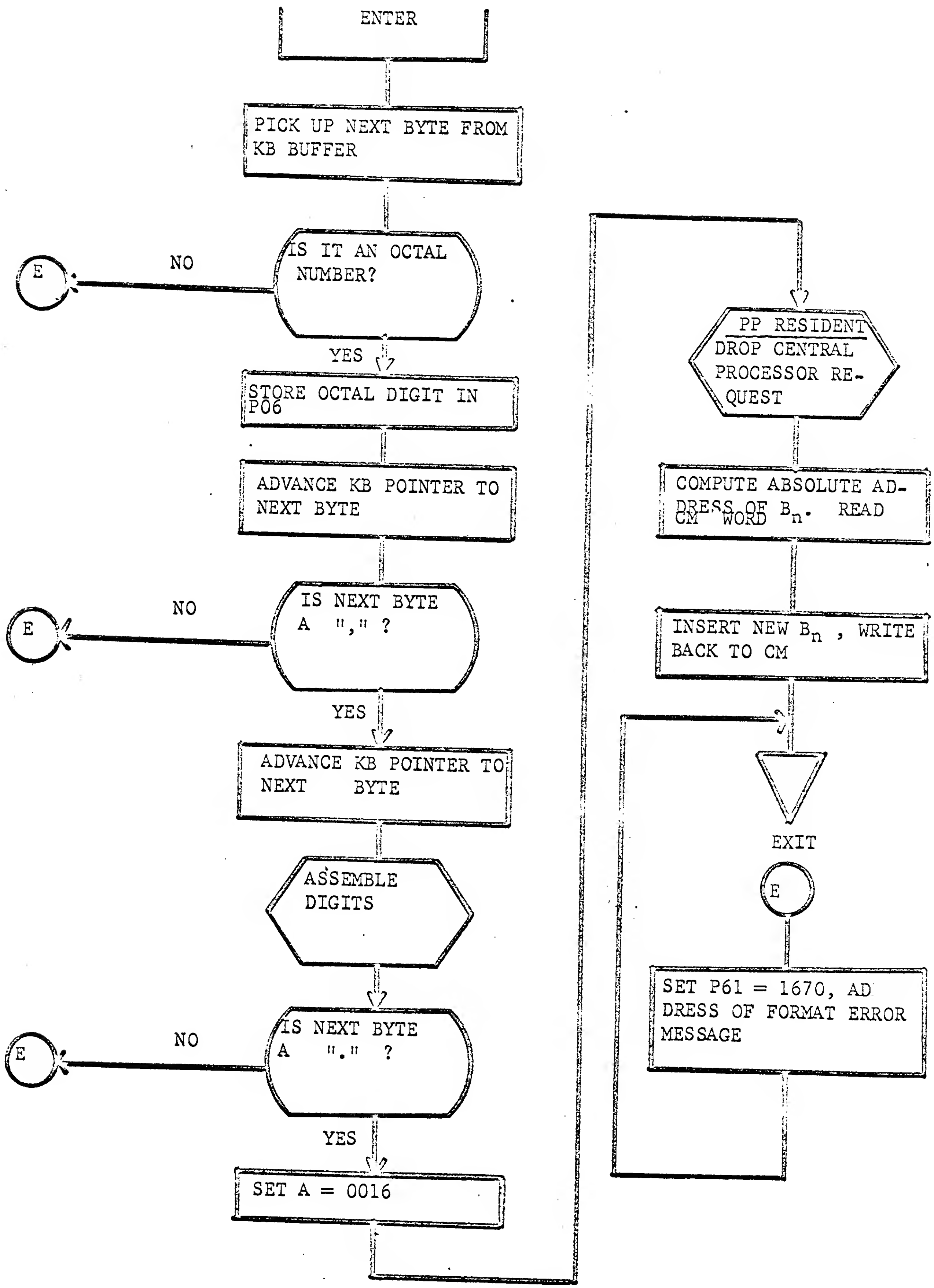
DIS - ENA REQUEST PROCESSOR



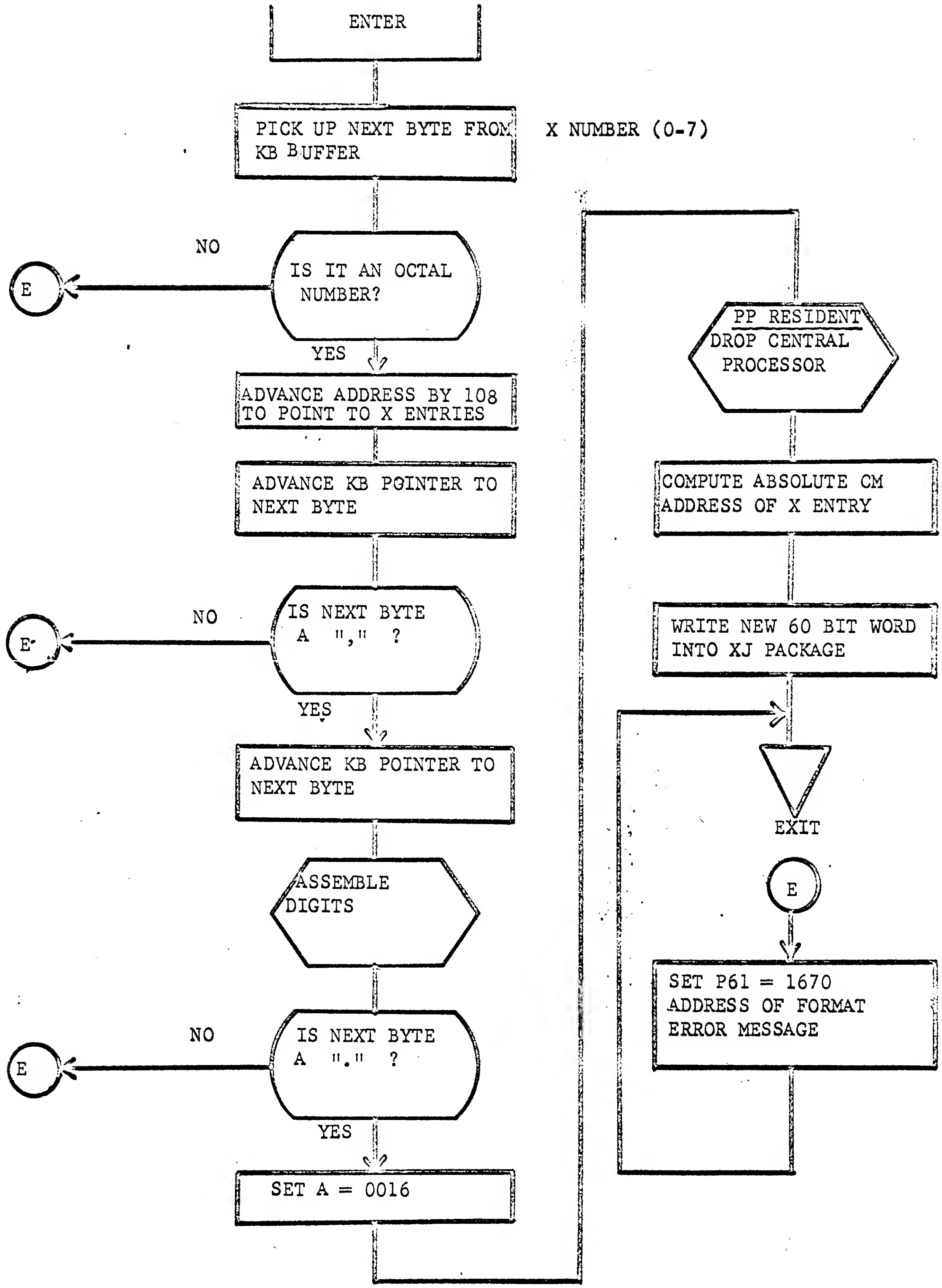
DIS - ENA REQUEST PROCESSOR (CONTINUED)



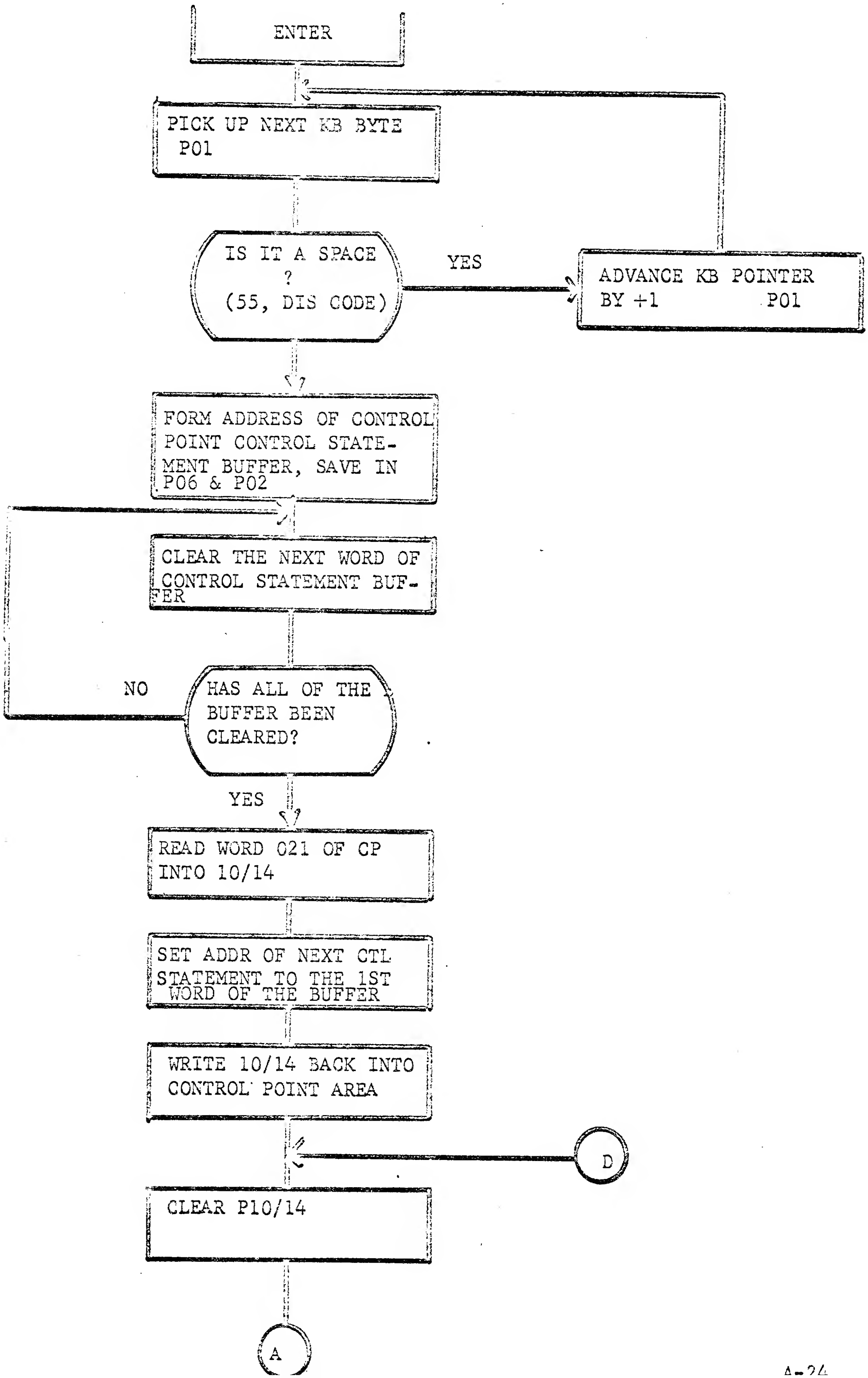
DIS - ENB REQUEST PROCESSOR



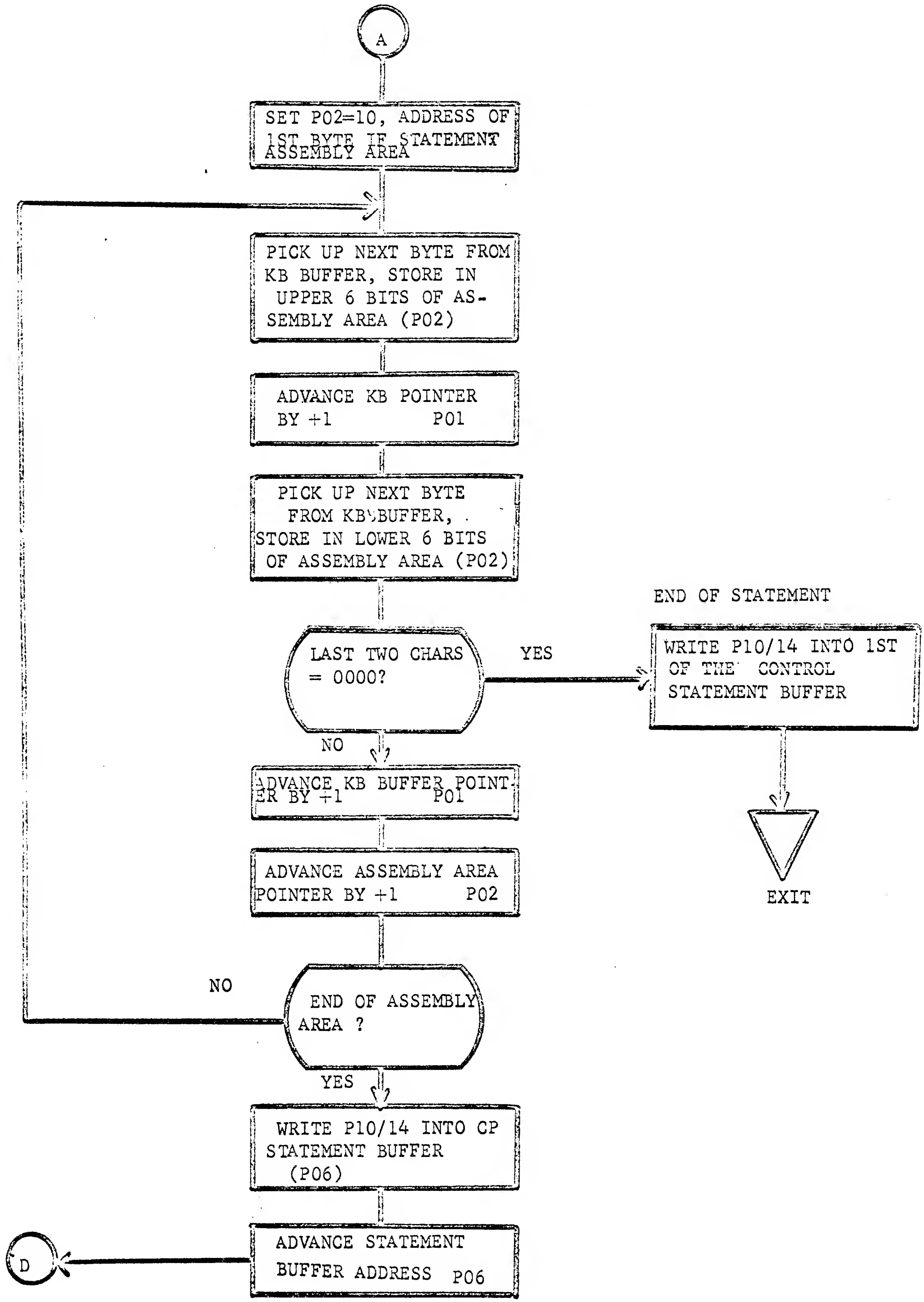
DIS - ENX REQUEST PROCESSOR



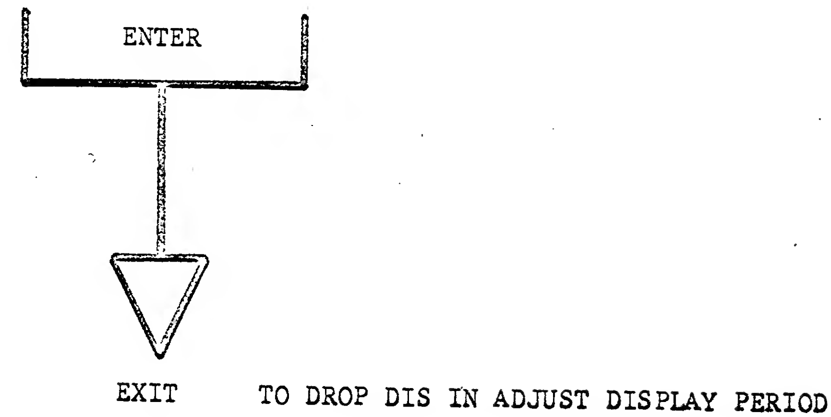
## DIS - ENS REQUEST PROCESSOR



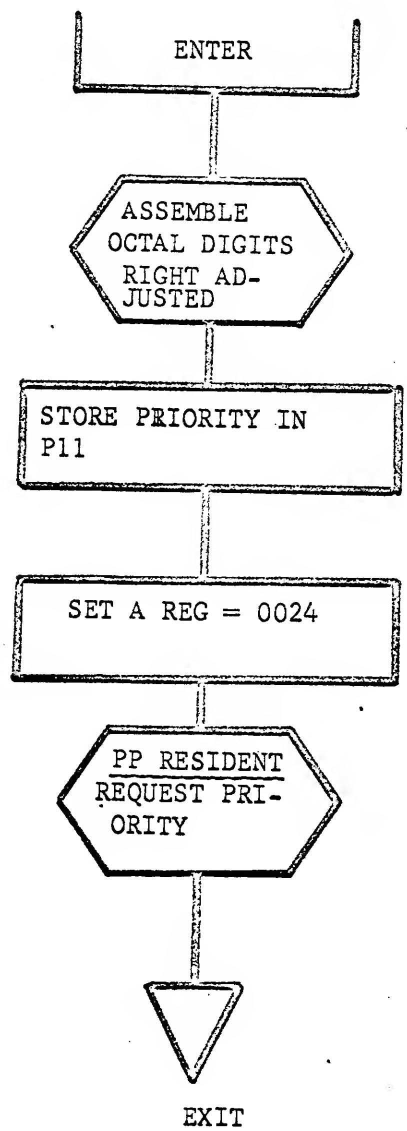
DIS - ENS REQUEST PROCESSOR (CONTINUED)



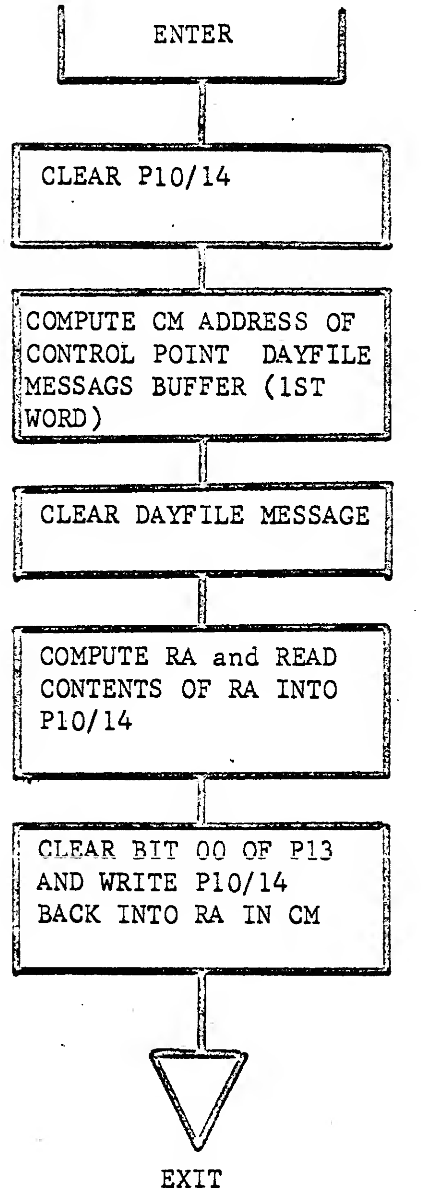
DIS - DROP REQUEST PROCESSOR



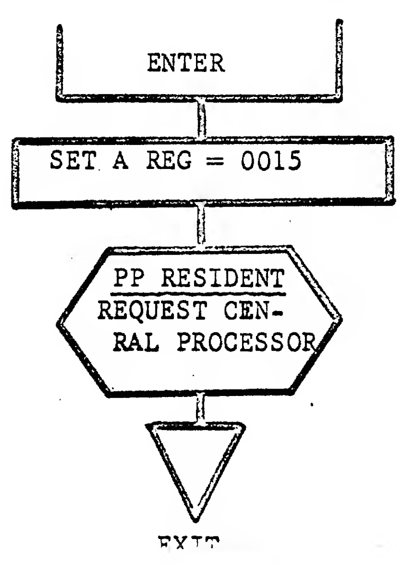

DIS - ENPR REQUEST PROCESSOR



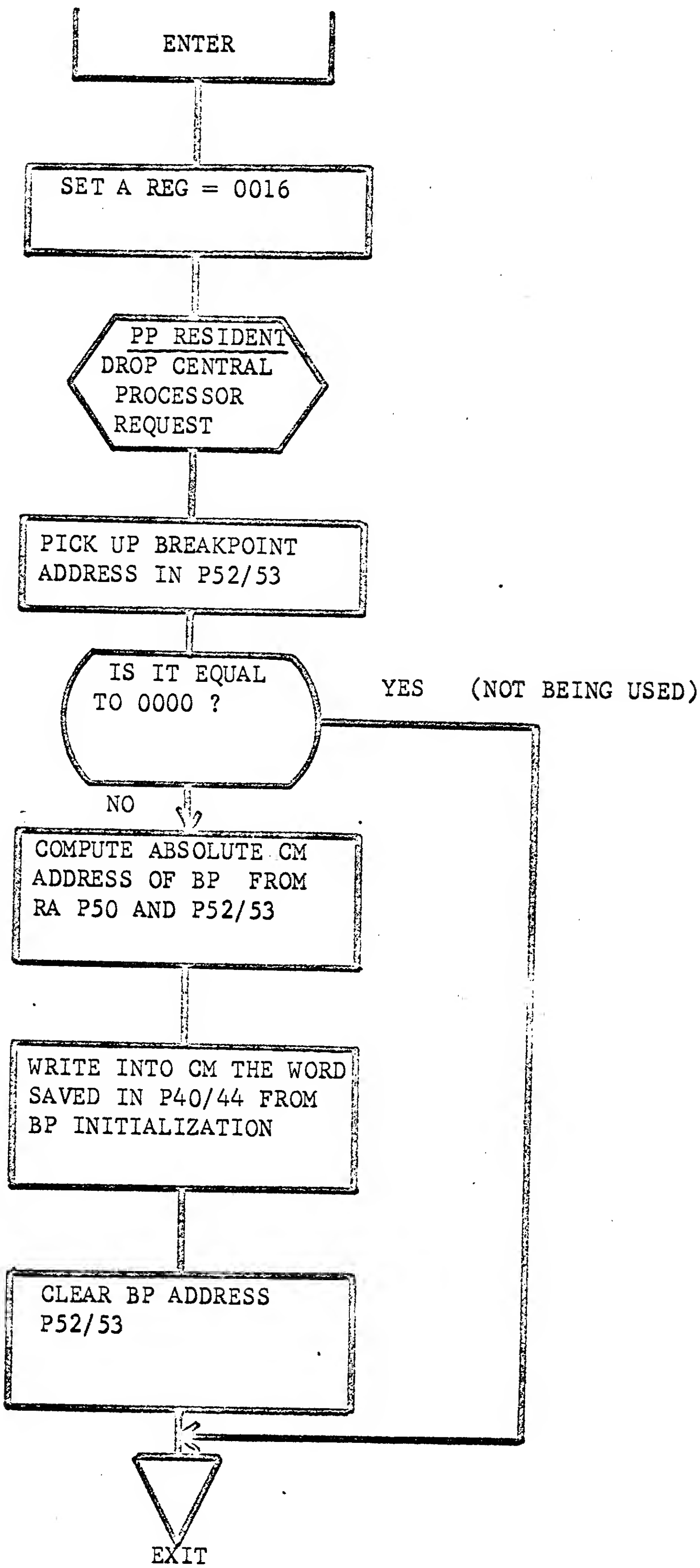
DIS - GO REQUEST PROCESSOR



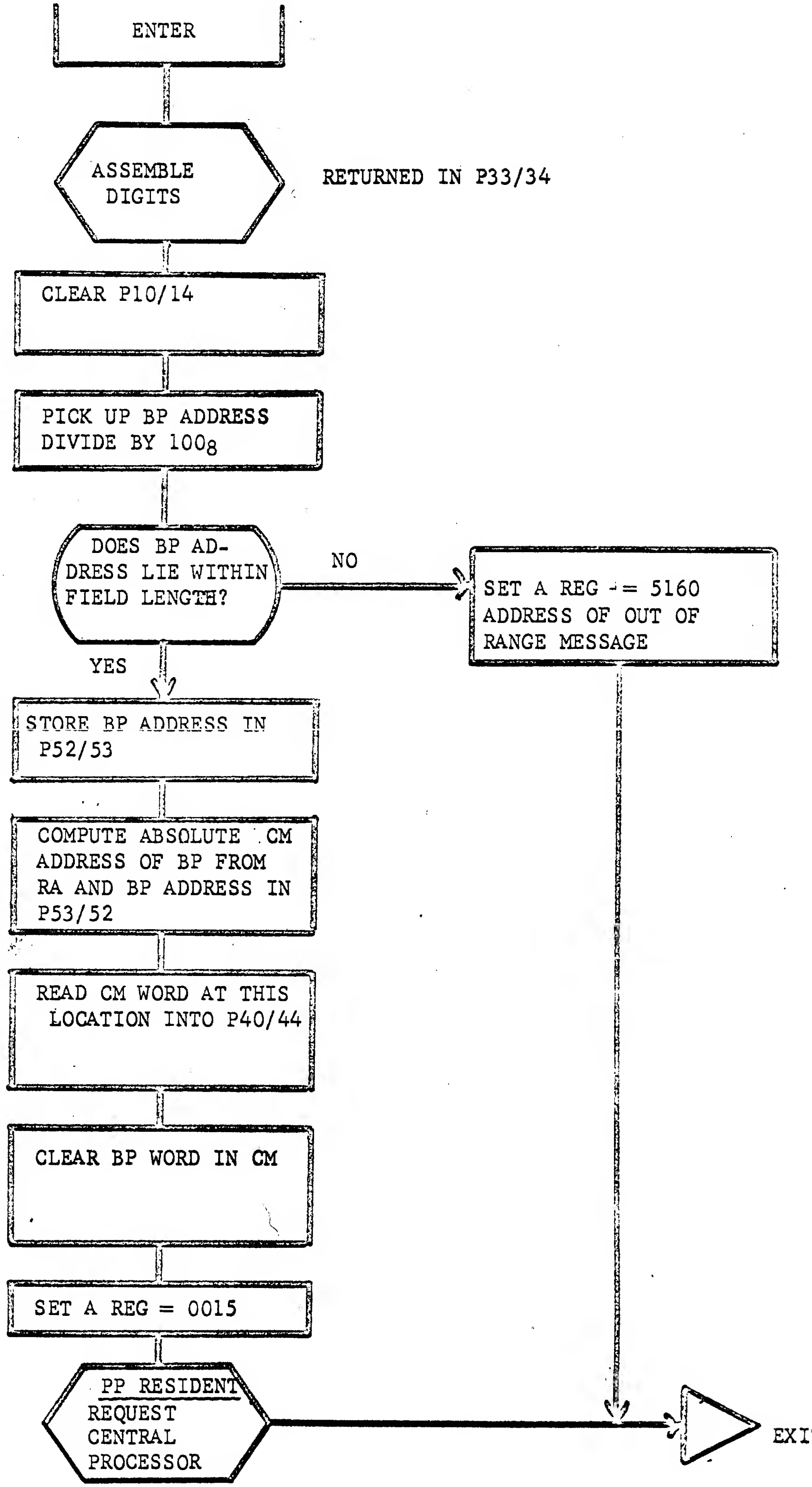

DIS - RCP REQUEST PROCESSOR



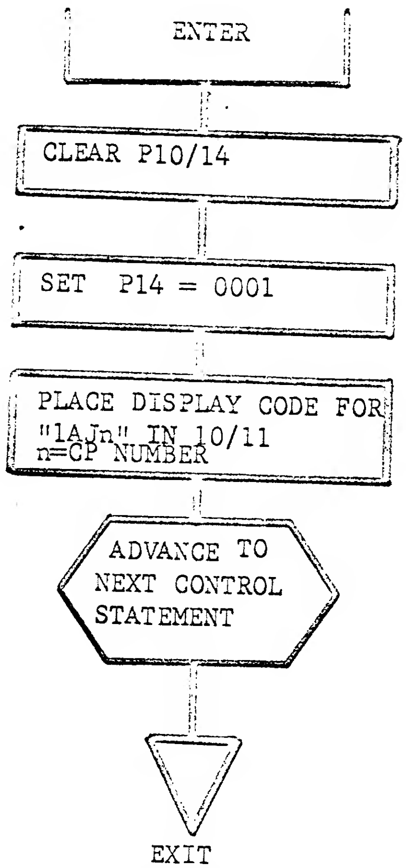
DIS - DCP REQUEST PROCESSOR



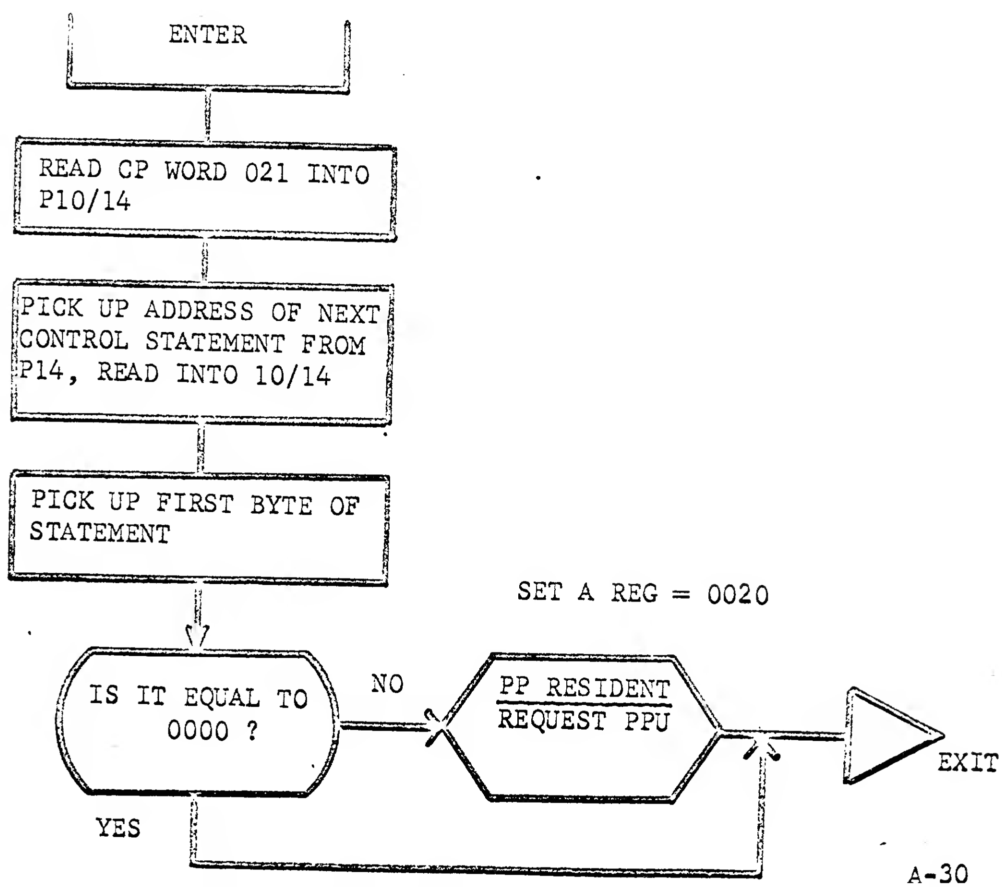
DIS - INITIATE BREAKPOINT REQUEST PROCESSOR



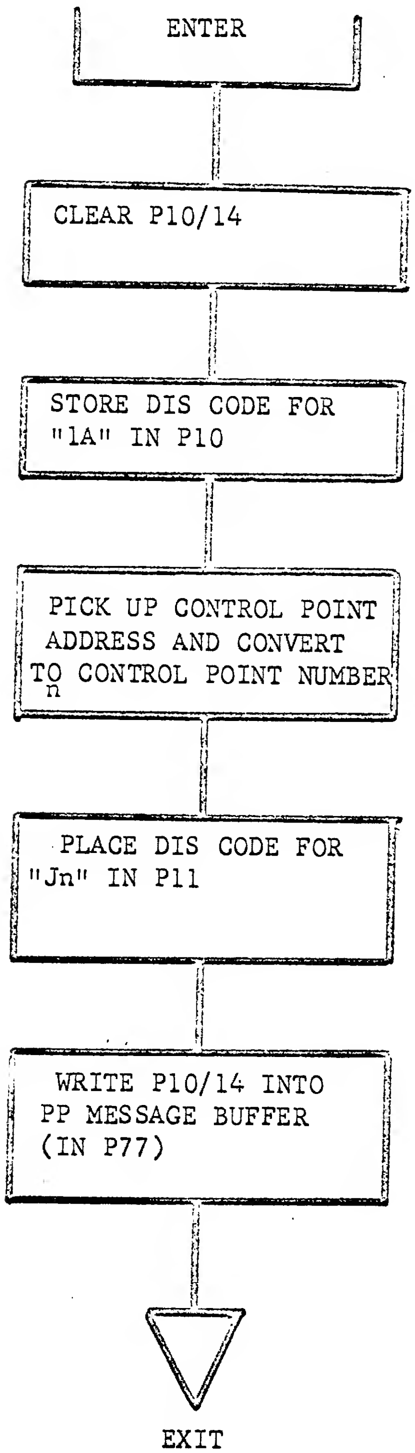
## DIS - RSS REQUEST PROCESSOR



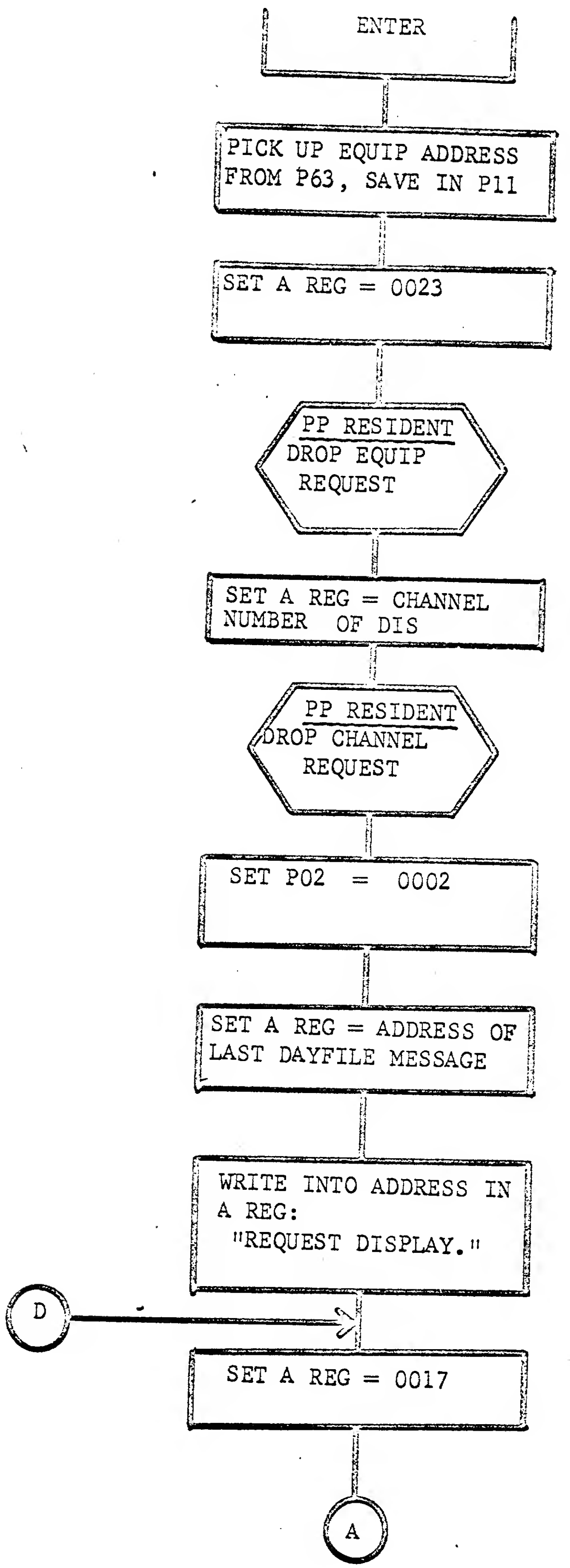

## DIS - ADVANCE TO NEXT CONTROL STATEMENT

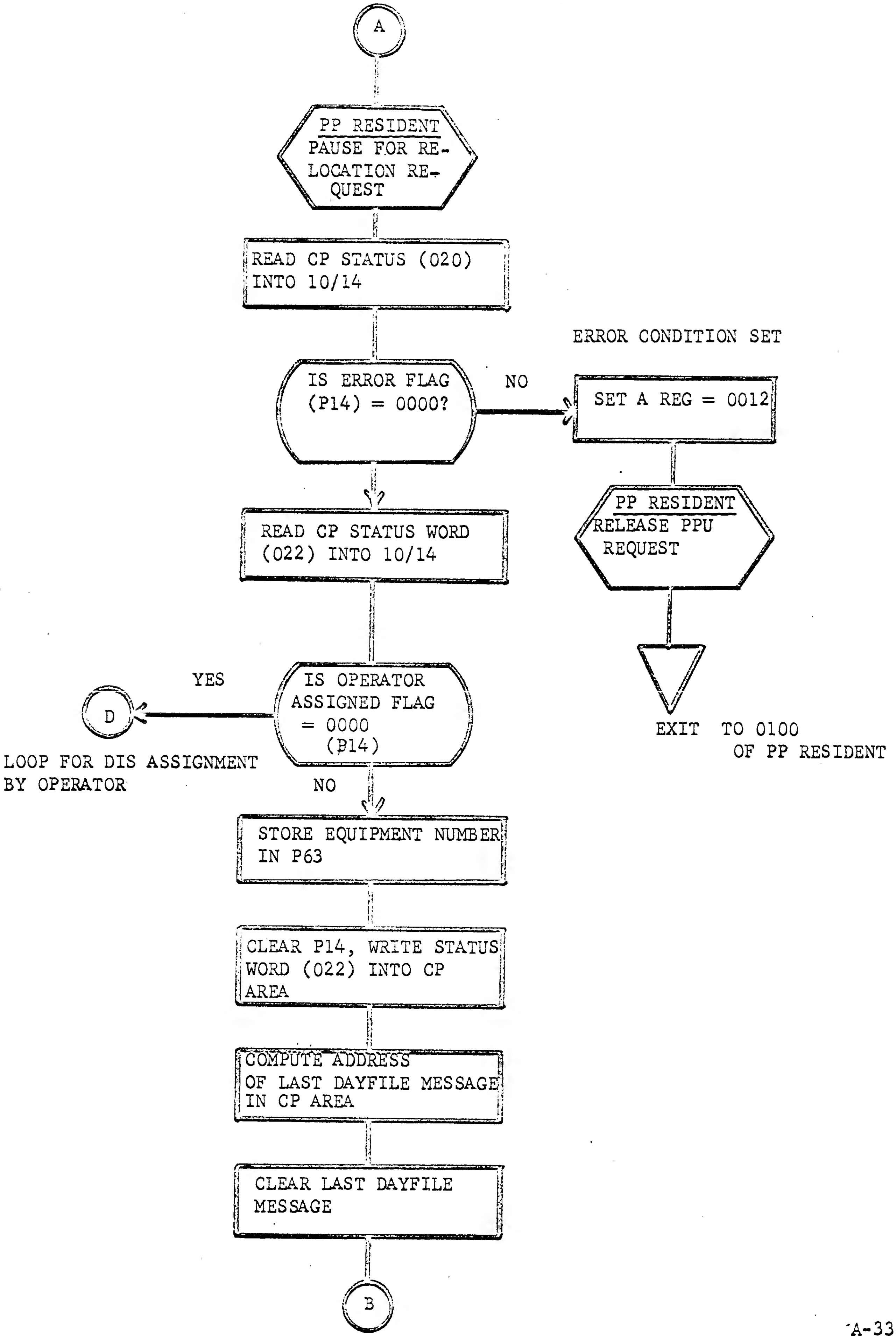


## DIS - RNS REQUEST PROCESSOR

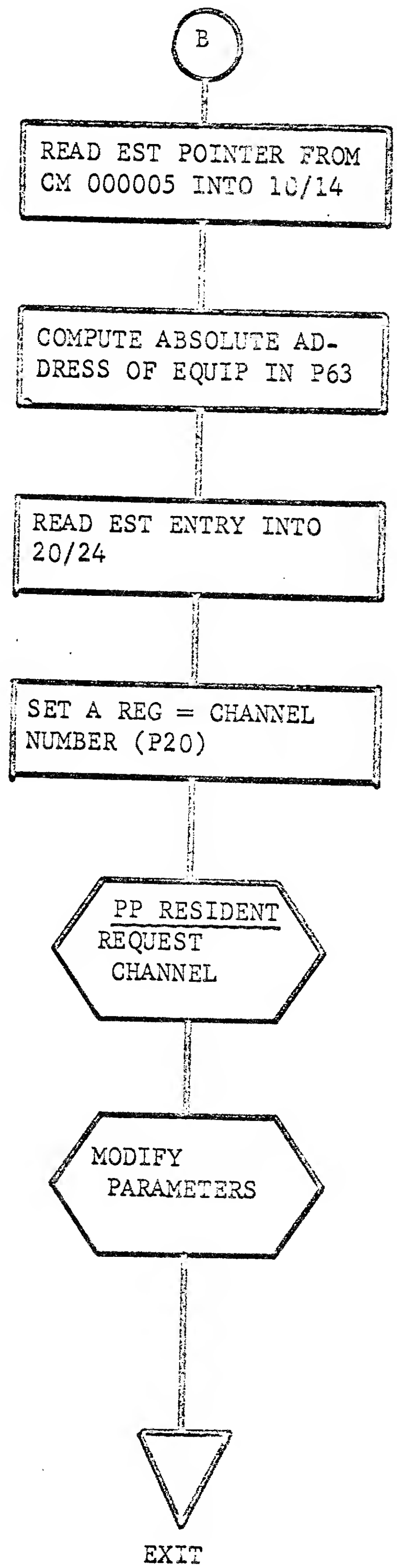


DIS - HOLD REQUEST PROCESSOR

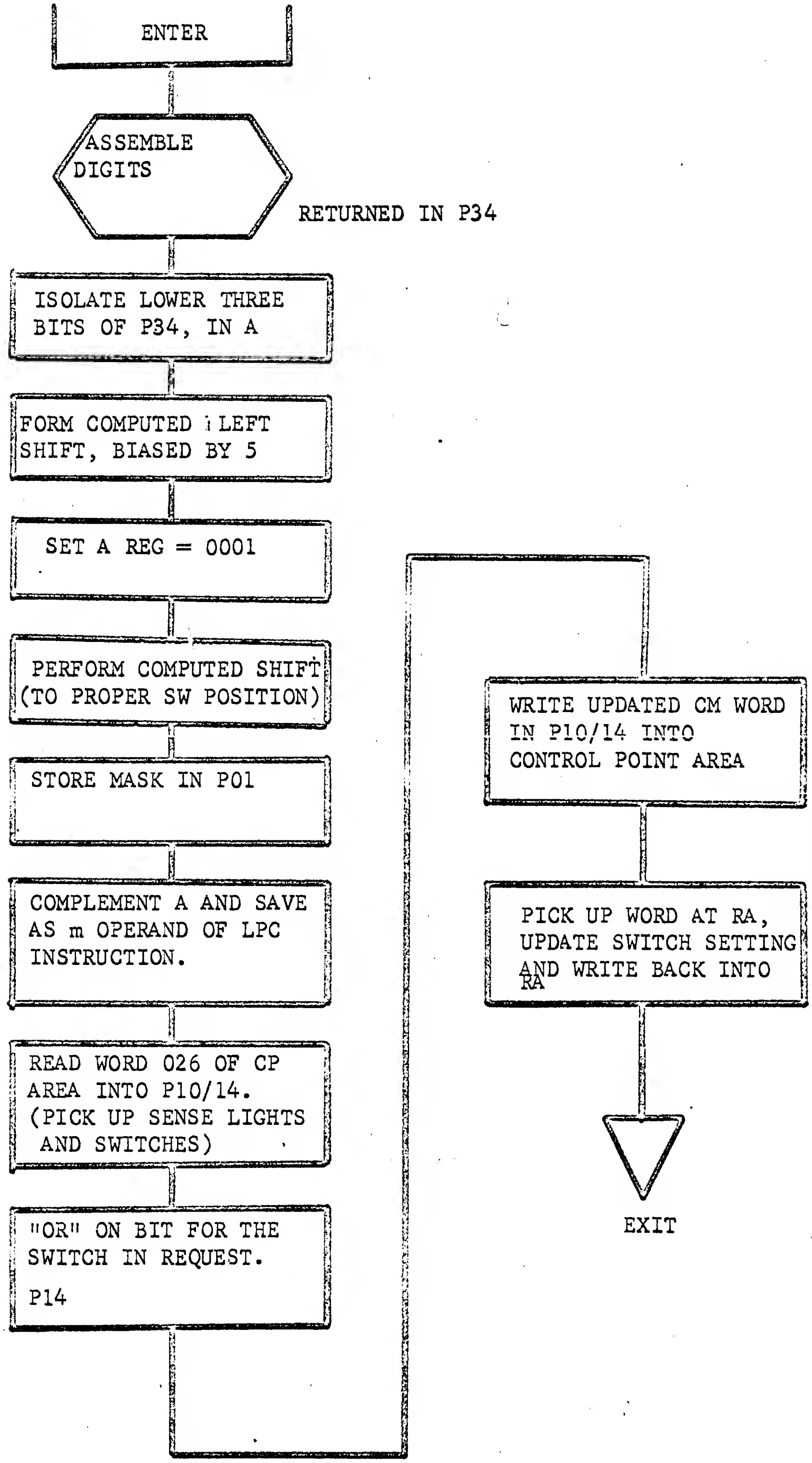



A
PP RESIDENT
PAUSE FOR RE-LOCATION RE-QUEST
READ CP STATUS (020) INTO 10/14
ERROR CONDITION SET
IS ERROR FLAG (P14) = 0000?
NO
SET A REG = 0012
PP RESIDENT
RELEASE PPU REQUEST
EXIT TO 0100 OF PP RESIDENT
READ CP STATUS WORD (022) INTO 10/14
YES
D
IS OPERATOR ASSIGNED FLAG = 0000 (P14)
LOOP FOR DIS ASSIGNMENT BY OPERATOR
NO
STORE EQUIPMENT NUMBER IN P63
CLEAR P14, WRITE STATUS WORD (022) INTO CP AREA
COMPUTE ADDRESS OF LAST DAYFILE MESSAGE IN CP AREA
CLEAR LAST DAYFILE MESSAGE
B

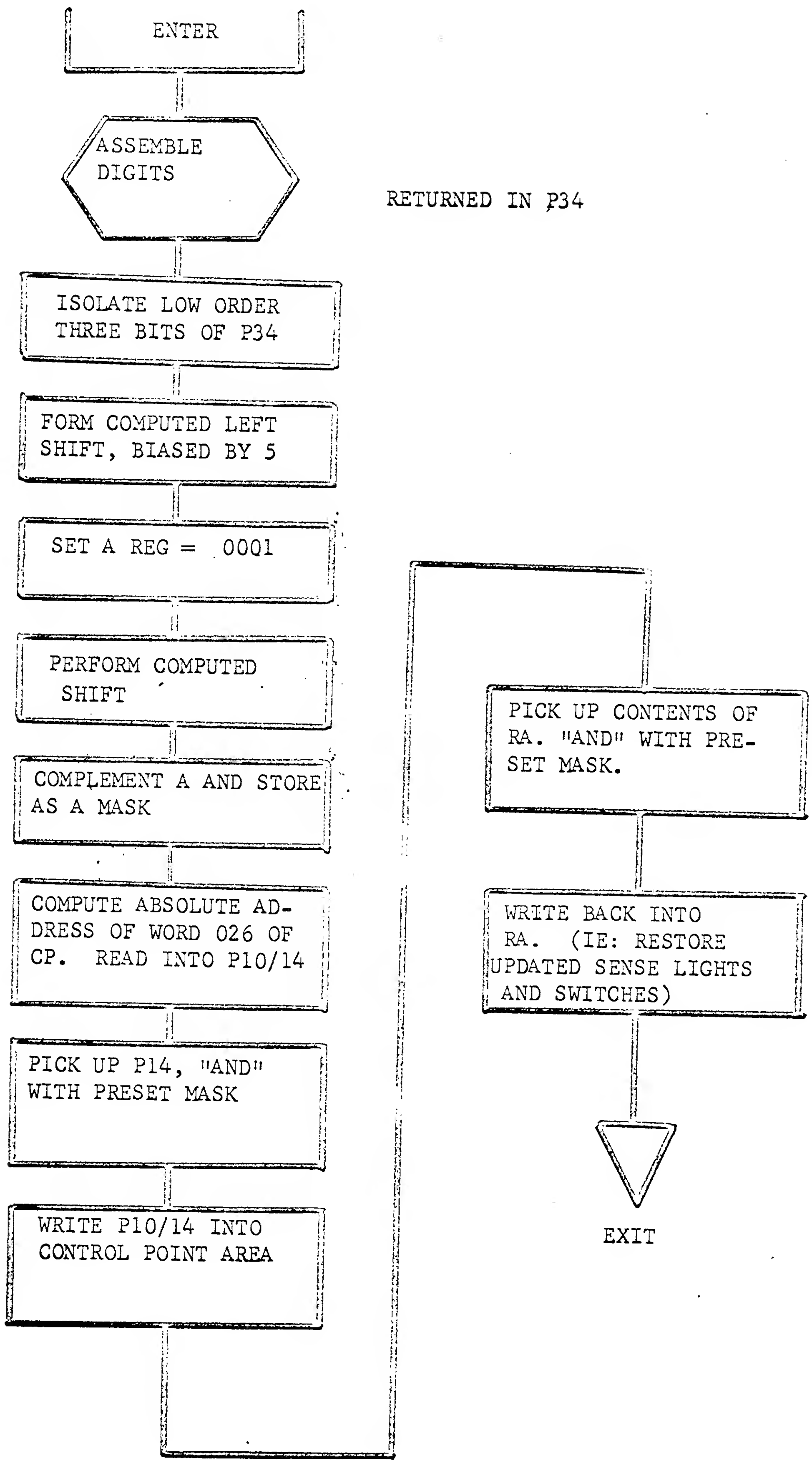
DIS - HOLD REQUEST PROCESSOR (CONTINUED)



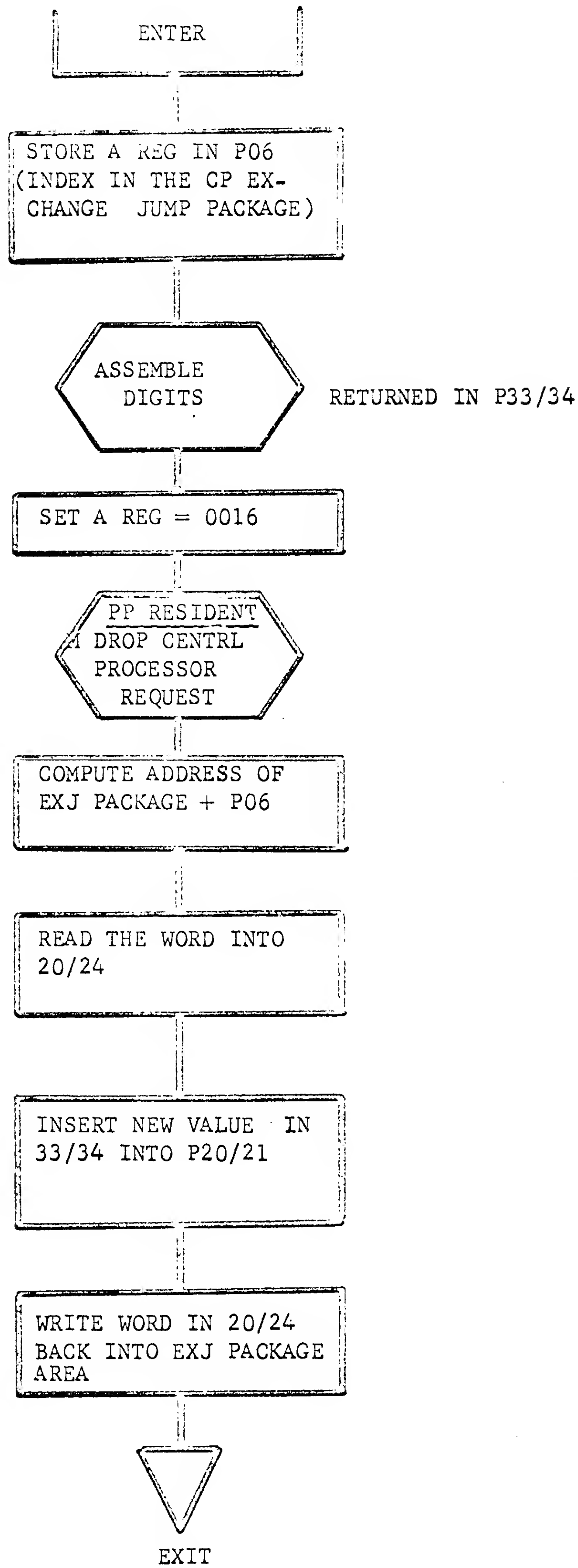
## DIS - ONSW REQUEST PROCESSOR



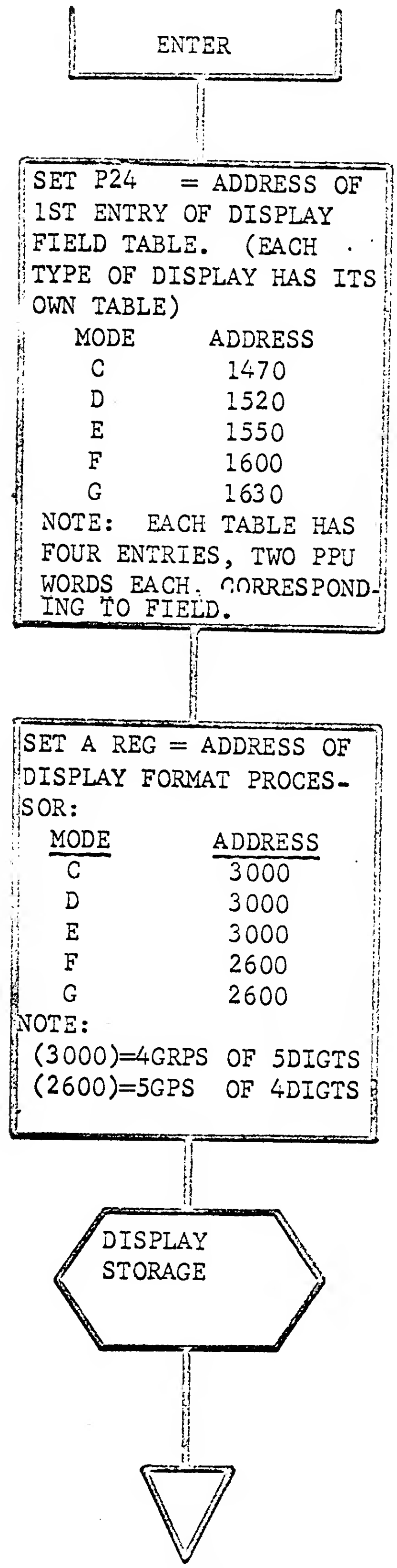
DIS - OFFSW REQUEST PROCESSOR



DIS - ENTER P, FL, RA, EM



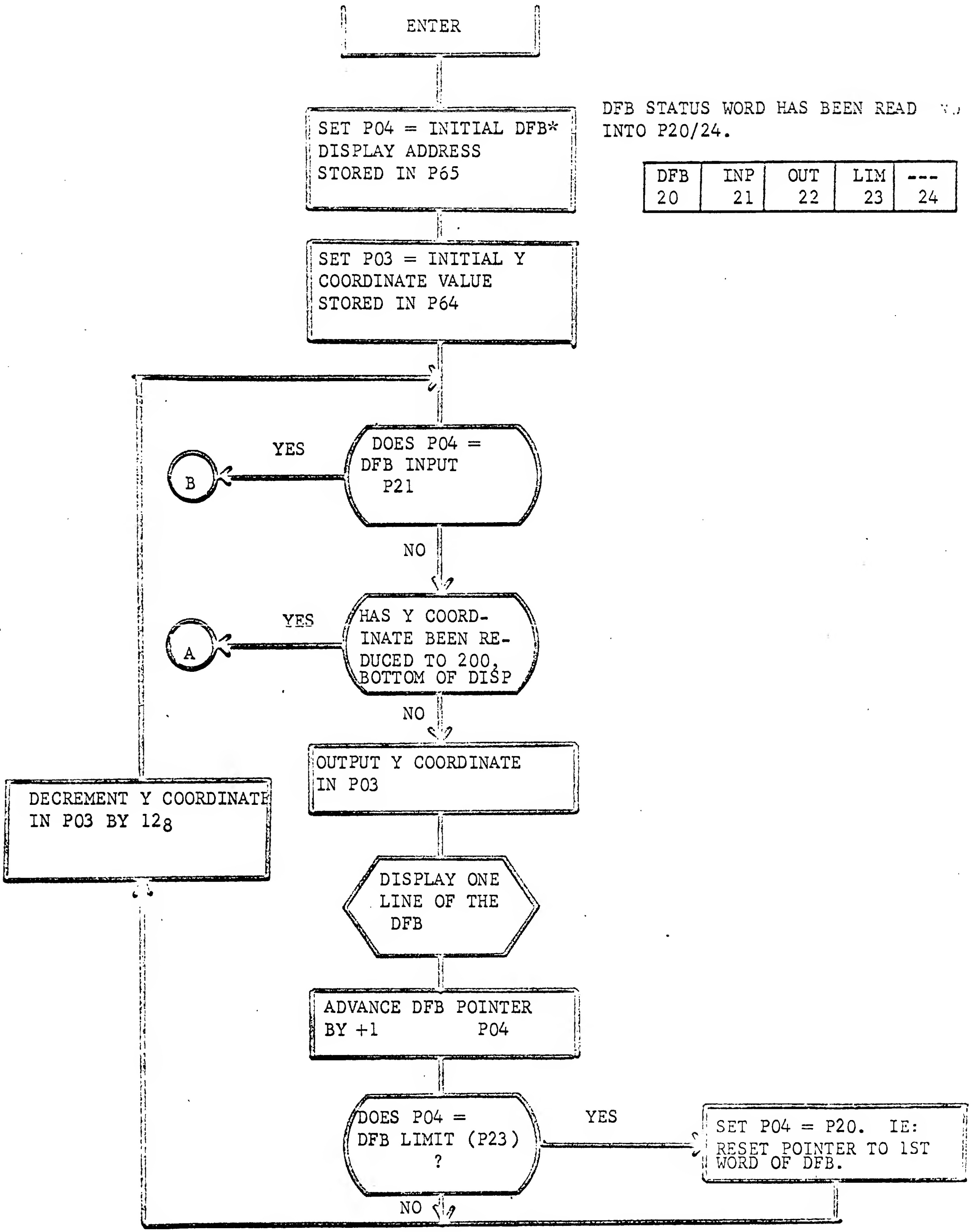
DIS - DISPLAY C, D, E, F, G



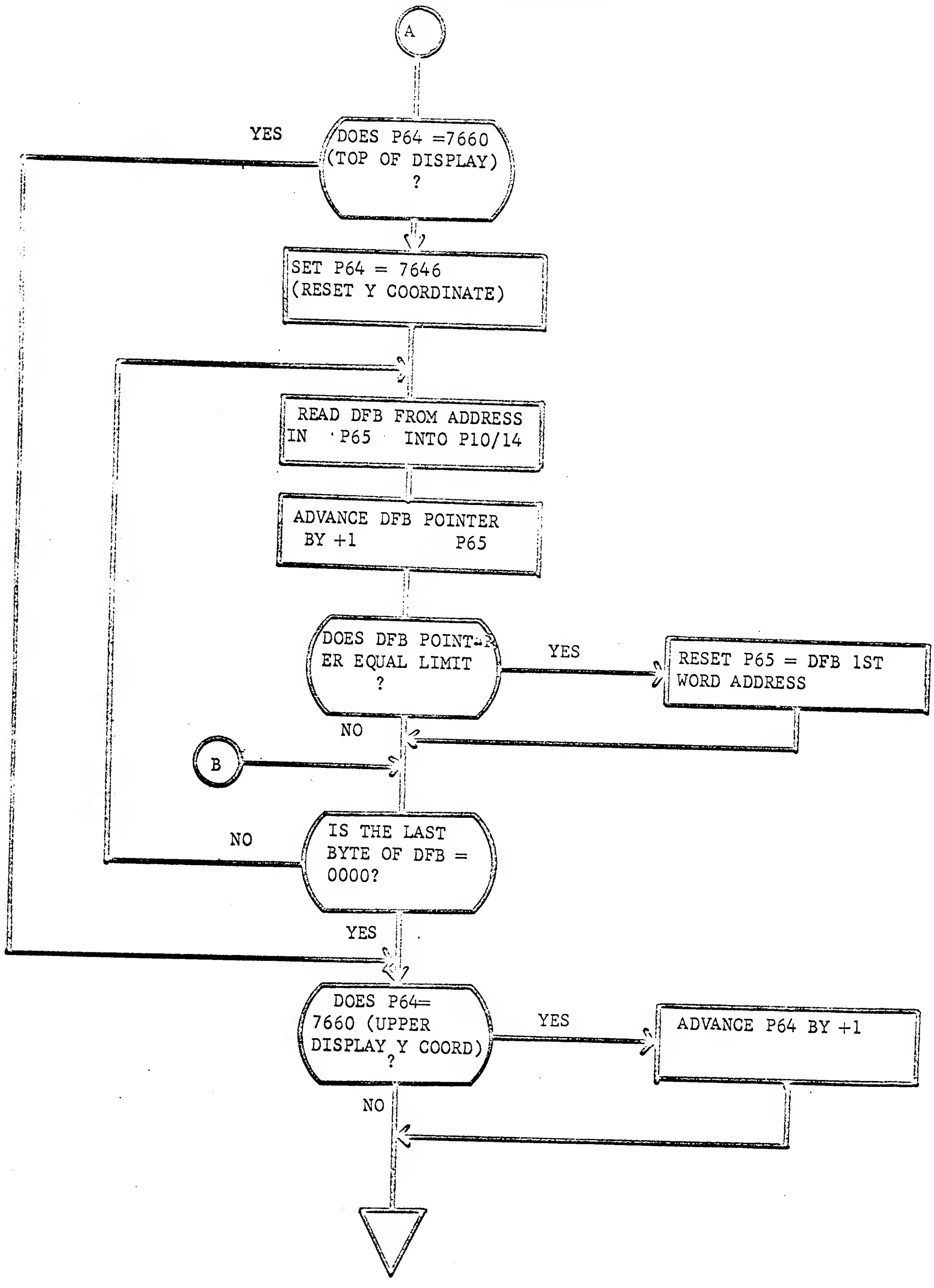
## DIS - DISPLAY DAYFILE

DFB STATUS WORD HAS BEEN READ INTO P20/24.

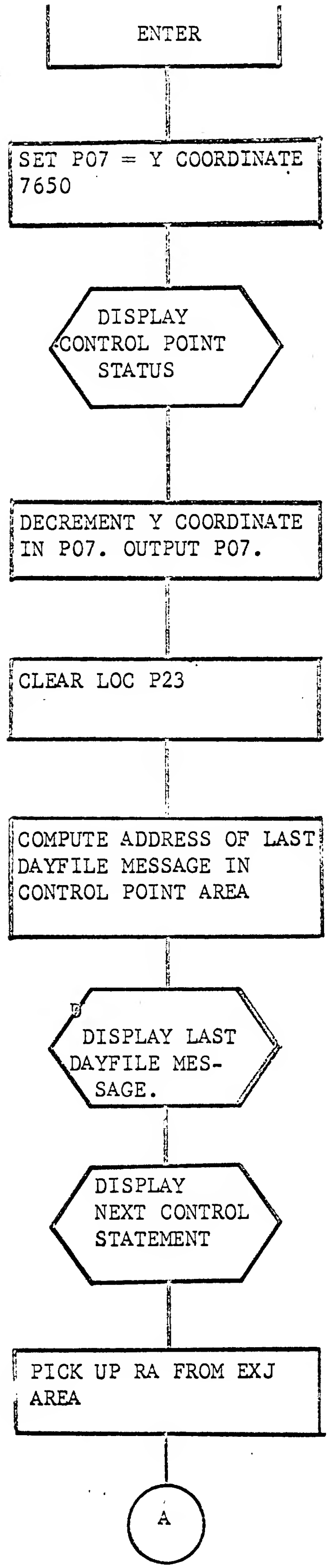
| DFB | INP | OUT | LIM | --- |
|---|---|---|---|---|
| 20 | 21 | 22 | 23 | 24 |



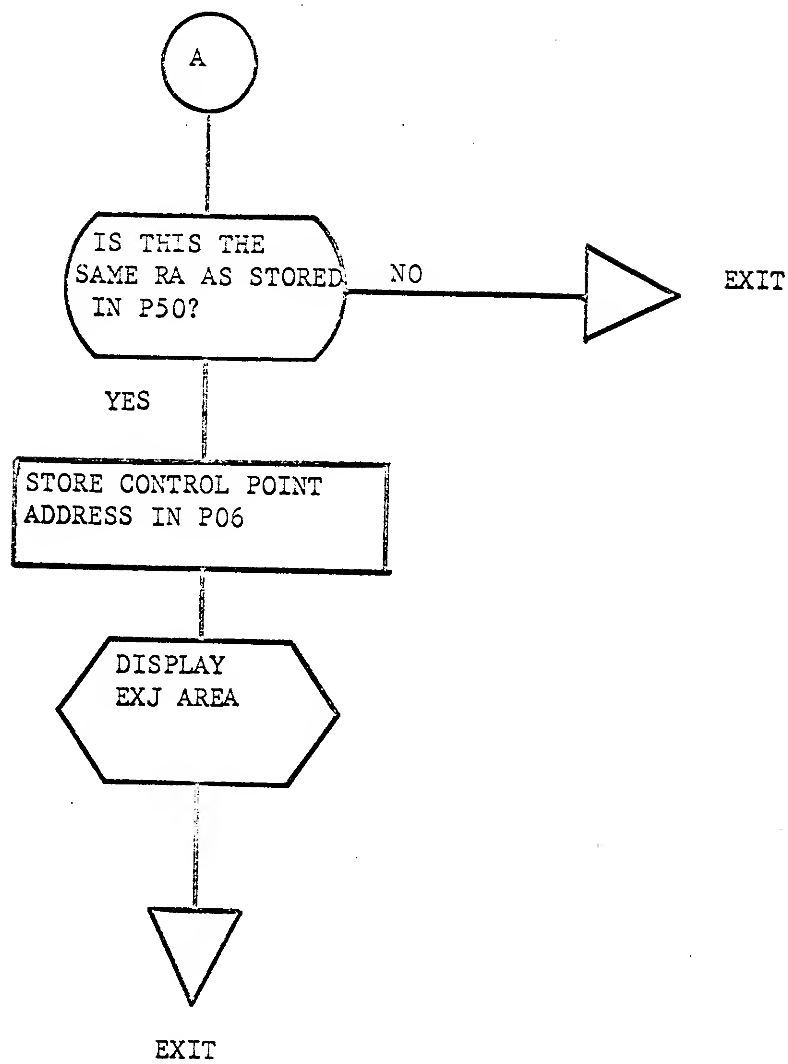
DIS - DISPLAY DAYFILE (CONTINUED)



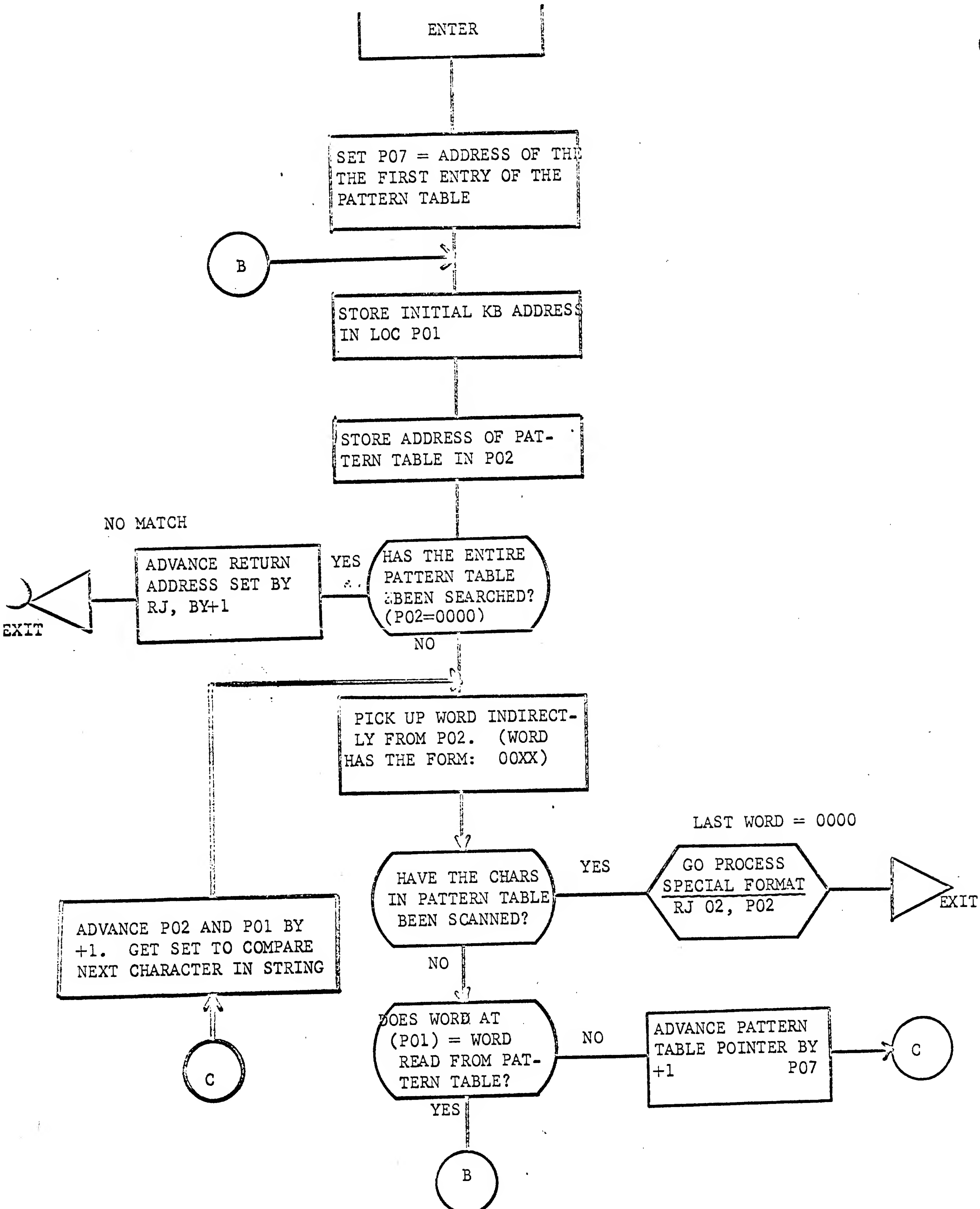
DIS - DISPLAY B



DIS - DISPLAY B (CONTINUED)



DIS - SEARCH FOR SPECIAL FORMAT



CONTROL DATA CORPORATION

Development Division - Applications

# DISK ROUTINES AND OVERLAYS

Chippewa Operating System

# Disk Routines and Overlays

## Contents

# DISK ROUTINES AND OVERLAYS

## Introduction

In the Chippewa Operating System, there is no single system element used to perform disk operations for all other elements of the system. Instead, each system element performs its own disk operations. This, while requiring additional coding for each of the system elements using the disk, eliminates the need for a request queueing and priority scheme required by the use of a single system element to process all disk operations. In addition, the housekeeping required by a disk subroutine in one system element can overlap, to some extent, a disk operation being performed by another system element. Among the system elements which perform disk operations are:

- peripheral processor resident (reads transient programs from the disk library)
- MTR (writes the contents of the dayfile buffer to the disk)
- some transient programs (read overlays from the disk)

Disk operations for external users are performed via the overlays 2WD (write disk), 2RD (read disk), and 2BD (backspace disk). These overlays are called by CIO when a disk operation is requested by a central processor program. In addition, these overlays are used by certain transient programs to perform disk operations. Thus, 1LJ and 1LT call 2WD when loading jobs from the card reader and a tape unit, respectively, while 1DJ and 1TD call 2RD when transferring job output to the printer or a tape unit.

Regardless of where in the system they are performed, disk operations are similar: this discussion will therefore be limited to the overlays 2WD, 2RD, and 2BD. Before discussing these routines a short review of the physical characteristics of the 6603 disk file is in order.

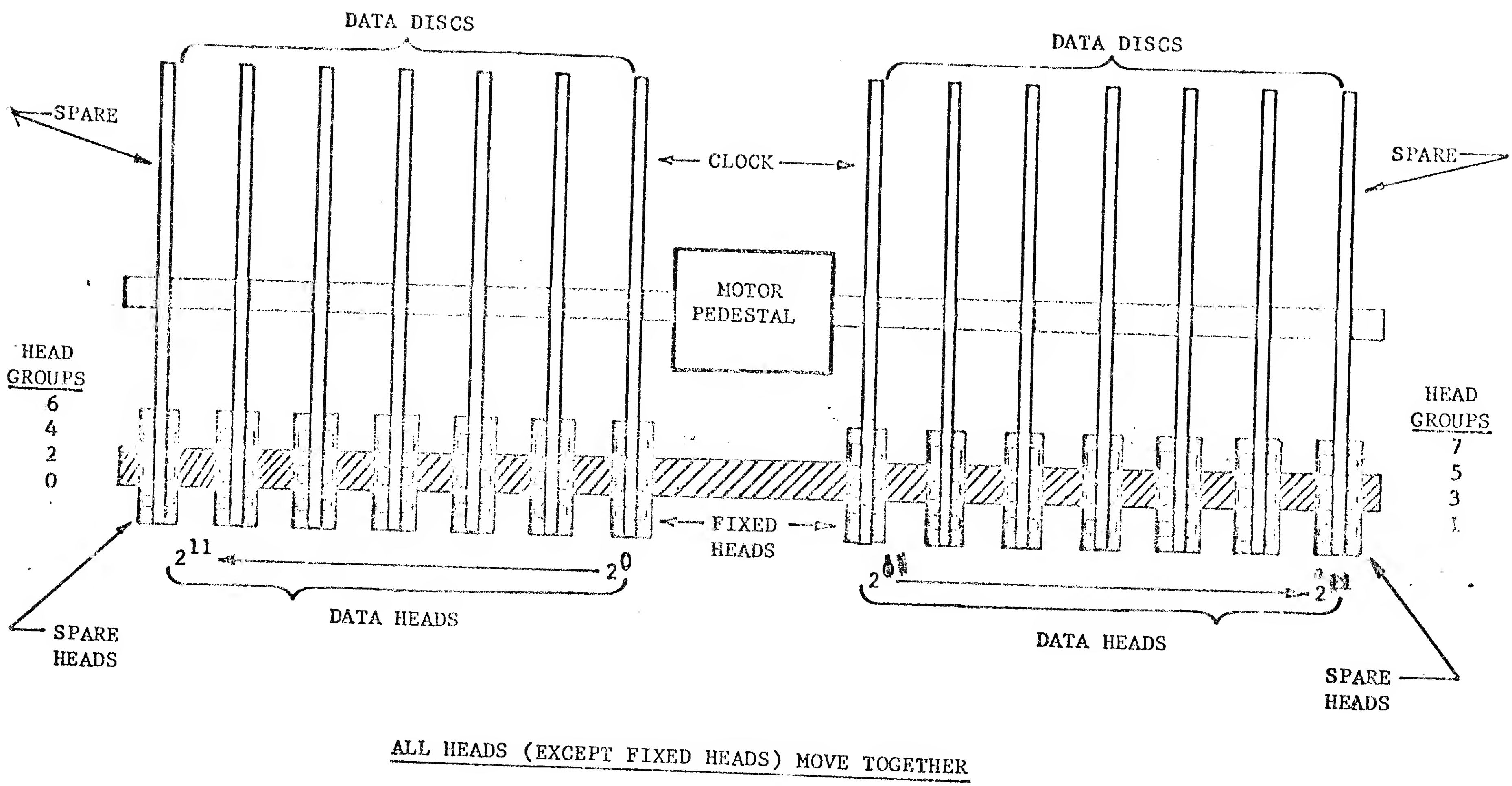
6603 Disk File: Description and Organization

The 6603 Disk File contains fourteen disks, each coated on both sides with magnetic oxide. Thus, there are a total of twenty-eight recording surfaces. On two of these surfaces timing tracks are recorded, two are used for spares, and twenty-four are used for recording data (see figure 1). All fourteen disks are mounted (in a vertical plane) on a common axis and rotate at a speed of approximately 900 revolutions per minute. Twelve of the data surfaces are on the right side of the unit, and twelve are on the left. Information is recorded on the disk in 12-bit bytes: each bit in a 12-bit byte is recorded on a separate disk surface.

Associated with each disk surface is a set of four read/write heads (see figure 2). An assembly consisting of a rocker arm and a head bar fits between each pair of facing disk surfaces. The head bar holds two sets of four heads, one set for each of the two facing surfaces. The read/write heads are mounted on this head bar in a fixed position relative to each other. The rocker arm-head bar assemblies for all disks mount on a common bracket which can be rotated. This rotation moves all the head bars simultaneously (with the exception of the heads accessing the timing track surfaces: these heads are fixed).

The disk surface is divided into four zones. A zone is that portion of the disk surface transversed by one of the four heads associated with that surface as the head (on its head bar-rocker arm assembly) moves through its maximum angular rotation. A byte may be written on the twelve data surfaces on the right side of the disk file or on the twelve data surfaces on the left side of the disk file: on either side, a byte may be written in any one of four zones. On each side of the disk file and for each zone on side, a single set of twelve read/write heads are used to record a byte (see figure 1). This set of twelve heads is called a head group. There are four head groups for each of the two sets of twelve disk surfaces: a total of eight head groups.

Each zone contains 128 tracks. A track is the recording path available to a given head group in a given position as the disk makes a complete revolution. To move from one track to another requires a physical



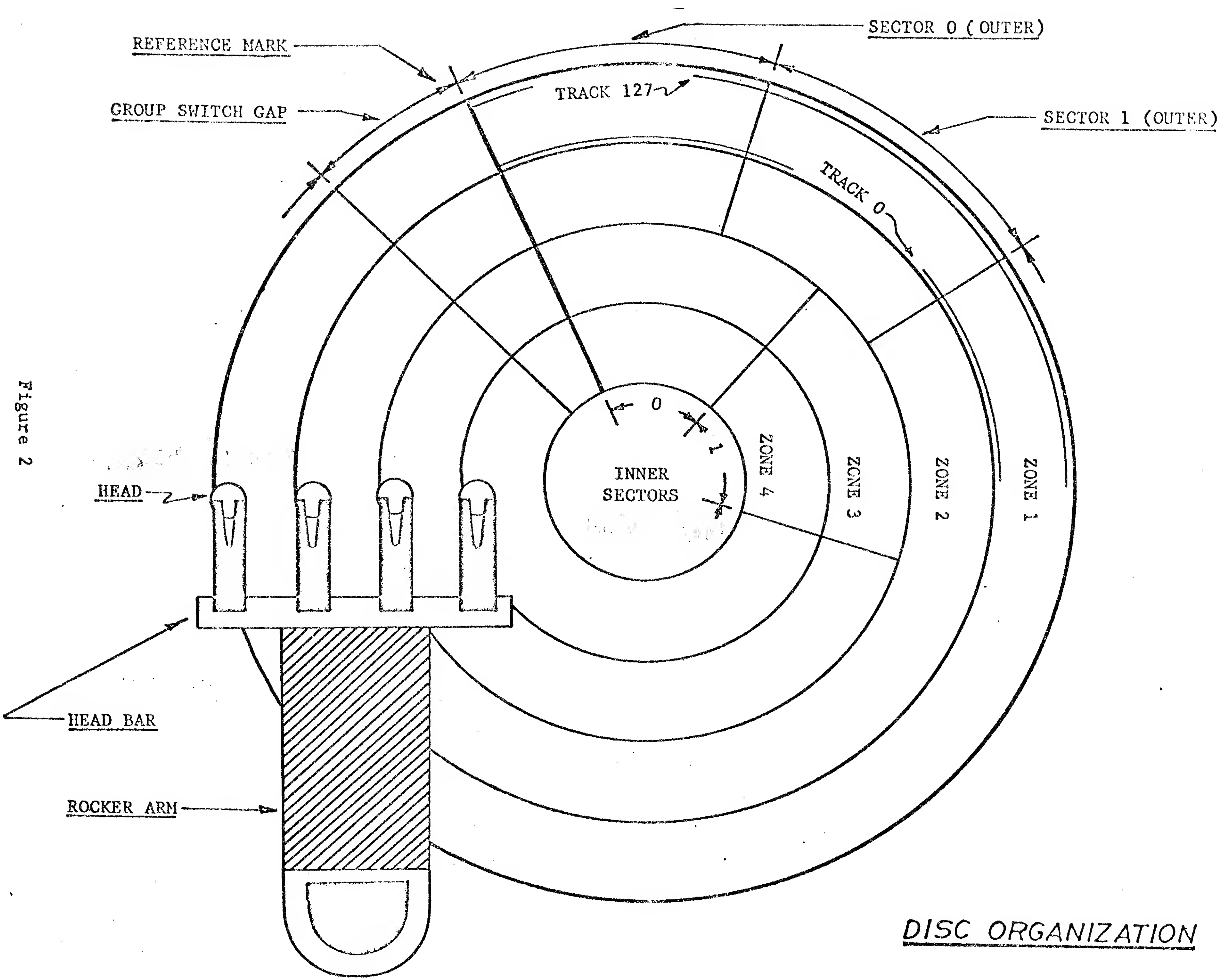

6603 DISC FILE

movement, or repositioning, of the head bar-rocker arm assemblies. At a given position, each head group accesses the same track in its zone. Thus, if head group 2 is positioned to track 125, the other 7 head groups are also positioned to track 125.

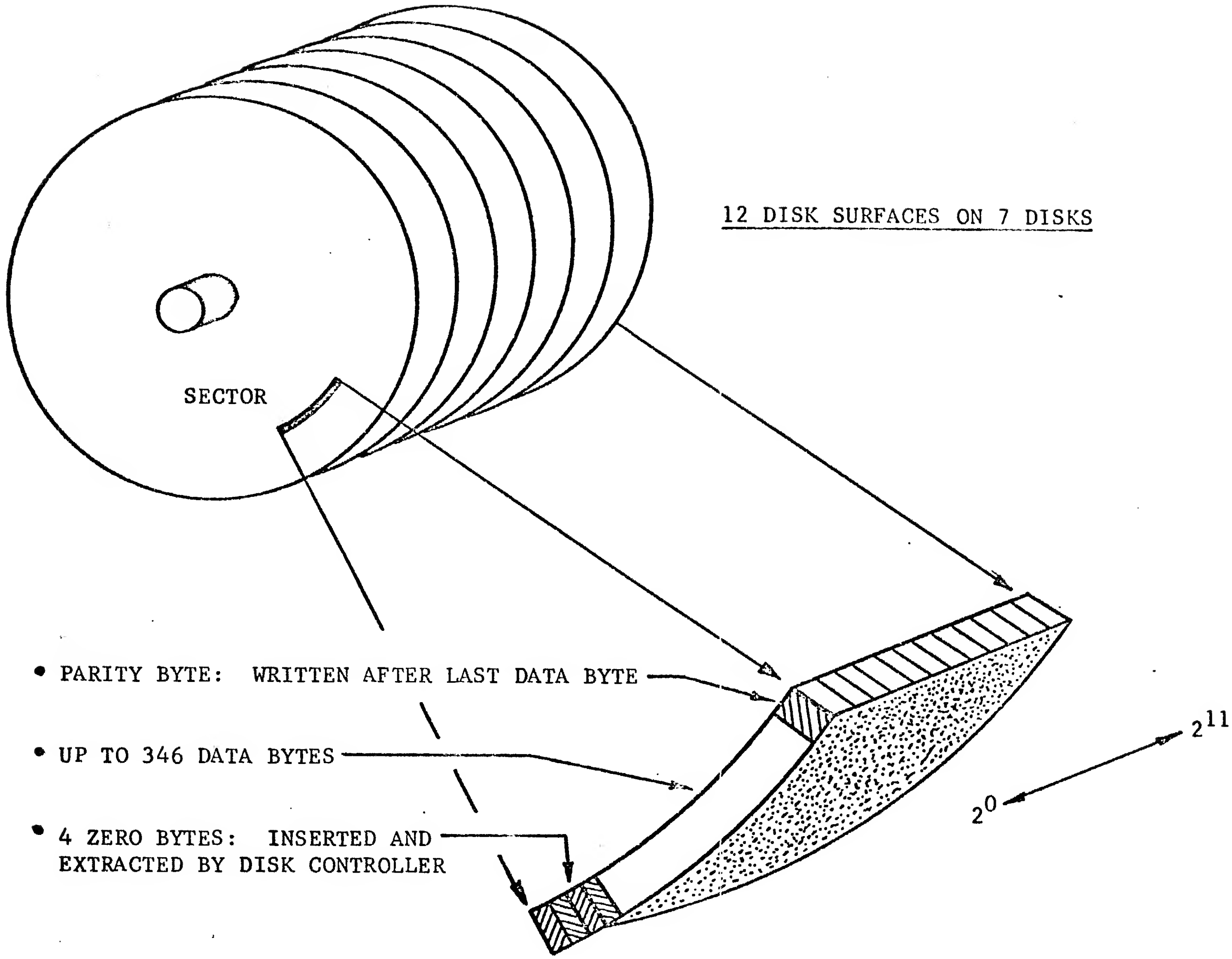
Tracks are divided into sectors: a sector is the smallest addressable segment of a track. There are 128 sectors in each of the tracks in the two outer zones. In the two innermost zones, there are only 100 sectors per track because of the reduced track length near the center of the disk compared to the track length available near the outside edge. A sector contains 351 bytes (each bit in a byte is recorded in one of 12 corresponding sectors across 12 disk surfaces). The first four bytes recorded are reserved for use by the controller: They provide a time lag between consecutive sectors and contain all zero bits. After the last data byte has been written, the controller writes a longitudinal parity byte. . The sector format is illustrated in figure 3. Of the 351 bytes in a sector, then, five are used by the controller: The remaining 346 bytes may be used for data. Normally, 320 bytes (the equivalent of 64 central memory words) are used for data.

The number of words read from or written to the disk is solely a function of the word count specified in the IAM or OAM instruction. It is possible to read or write more than one sector at a time; it is possible to read or write in the group switch gap; it is possible for a read or write to wrap around on the same track. A read or write operation always begins at the beginning of a sector. When a write is initiated, the disk controller inserts four zero bytes before the data and inserts a parity byte after the last data byte. (The parity byte is not necessarily in the last byte position in a sector.) When a read is initiated, the controller assumes that the first four bytes are zero bytes, and does not pass these on to the data channel. When the word count in a read has been reduced to zero, the controller assumes that the next byte to be read is the parity byte. Thus, any attempt to read a number of bytes different than the number of bytes written will invariably create problems due to the interpretation of zero bytes and parity bytes as data and vice versa. For this reason, regardless of the amount of data to be recorded, a fixed number of bytes is written in each sector,

Figure 2




DISC ORGANIZATION




SECTOR FORMAT: 6603 DISK FILE

Figure 3

and only one sector is written at a time (i.e., data is recorded in physical records of one sector).

A reference mark on the disks containing the timing tracks defines the beginning of sector 0 in all four zones. Beyond this point, the starting point of sectors in the two inner zones does not coincide with the starting point of sectors in the two outer zones (see figure 2). The clock surfaces contain timing tracks for each zone. As the disk rotates, one of these timing tracks (depending on which head group is selected) drives a cell counter. This counter in turn triggers a sector counter. Both counters are initialized when the reference mark is detected. The cell counter is incremented as the timing track is read: When it reaches a count of 351, it is reset and the sector count advanced. The controller compares the sector number specified in a read or write function code: When equality is obtained, the read or write operation is initiated. The contents of the sector counter appear in the low-order 7 bits of the status response.

## 6603 Disk File: Timing Considerations

The rotational speed of the disk is approximately 900 revolutions per minute, corresponding to a revolution time of about 66 milliseconds. The time required to read or write a byte is approximately 1.4 microseconds on the two outer zones and 1.8 microseconds on the two inner zones. In the outer zones, then, a sector passes under the heads every 490 microseconds. It requires a minimum of 325 microseconds to transfer the 64 central memory words in a sector from peripheral processor memory to central memory, and, because of memory and pyramid conflicts, will probably require longer. A single peripheral processor cannot maintain a continuous data flow between consecutive sectors on the disk and central memory.

If the programmer wishes to read or write in a given sector, he simply issues the appropriate function code and, when the sector comes under the heads, the operation is initiated. The programmer may prefer to minimize the time spent waiting for this sector by sensing (via a status request) the position of the disk. Timing considerations make

it impossible to sense for a given sector and then initiate an operation in that sector: If one wishes to read or write sector N, then sector N-2 should be sensed in order to assure that a revolution will not be lost.

There are two types of delays which are of concern to the disk programmer. One of these is the positioning delay: The time required to move the heads to a new track. When a track select function has been received by the disk controller and positioning initiated, a delay determined by counting ~~two~~ 4 reference marks is provided to permit the head assembly to stabilize. Thus, depending on when positioning is initiated, up to ~~133~~ 266 milliseconds may be required. During positioning, a status request will receive a "NOT READY" reply.

The second type of delay is the switching delay encountered when a different head group is selected. When head group switching is initiated, the controller provides a one millisecond delay to allow the circuits to stabilize: Furthermore, reading or writing cannot be initiated until a reference mark is detected. Thus, depending on when the head group select function is issued, up to 66 miliseconds may be required for head group selection.

Between the last sector in a track (sector 127 in the outer zones, sector 99 in the inner zones) and the first sector (sector 0) on that track is an area called the group switch gap (see figure 2). This area is approximately equivalent to three sectors in size. It is provided to accommodate the minimum 1 millisecond switching delay. A programmer can thus read or write the last sector in a track, select a new head group, and read or write sector zero of the new track without incurring a delay.

The function code for head group selection is 160X, where X is the head group number (0-7). It is possible to vary the second octal digit in this function code (normally zero) from 1 to 7: In doing so, the manner in which the data signals from the disk are sampled is varied. Use of the feature is reserved for error routines.

## 6603 Disk File: Data Capacity

There are 128 physical positions of the heads: At any one position, a track may be accessed by selecting one of eight head groups. Thus, the disk has a total of 8 x 128 = 1024 tracks. Of the eight head groups, four cover inner zones and four cover outer zones. In the inner zones, there are 100 sectors per track: In the outer zones, there are 128 sectors per track. Therefore, 512 tracks each contain 100 sectors while the other 512 tracks each contain 128 sectors. The disk file thus contains 116, 736 sectors. In normal use, up to 64 central memory words are recorded in a sector. The capacity of the 6603 disk file is thus approximately 7.5 million central memory words.

## Chippewa Operating System Disk Usage

As we have seen, a single peripheral processor cannot maintain a continuous data flow from consecutive disk sectors to central memory. Therefore, the Chippewa Operating System uses a half track scheme in its disk operations. A half track is composed of either the odd-numbered or the even-numbered sectors in a track. In a disk operation, the system reads or writes alternate sectors, transferring data to or from central memory while passing over the intervening sector. Since the disk contains 1024 physical tracks, the equivalent half track capacity is 2048. The allocation of half tracks is controlled by MTR: disk write routines obtain half track addresses from MTR via the Request Track function. MTR maintains a table called the Track Reservation Table (TRT) which contains an entry for each half track on a disk. On receipt of the Request Track function, MTR searches the table for an unassigned half track, and returns the half track address to the requestor in the upper byte of the Message Buffer. If no half track is available, a zero address is returned to the requestor. A half track is never split between files: thus, the half track is the smallest unit of storage allocated on the disk.

The format of the half track address, and its relationship to physical disk addresses, is illustrated below.

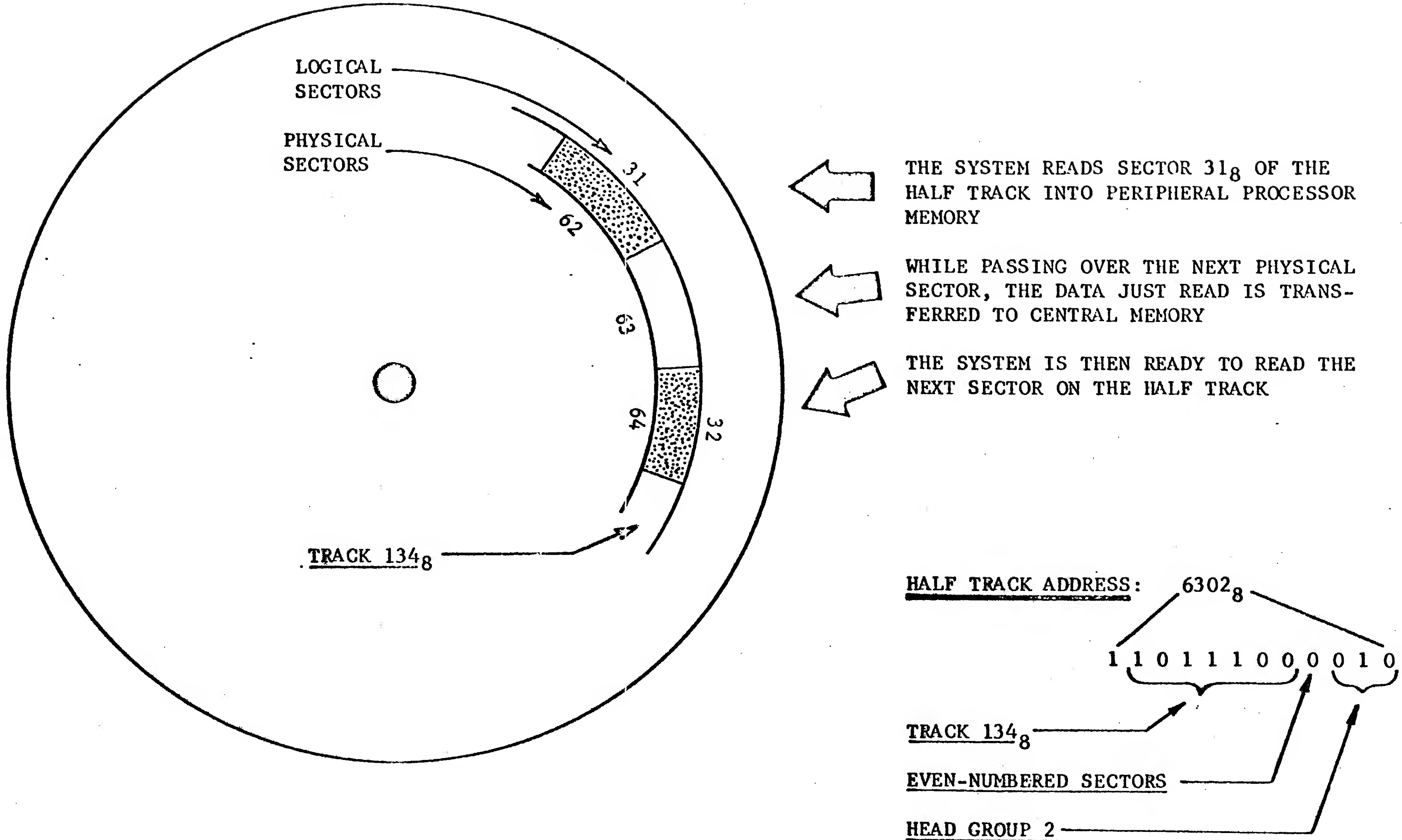



Figure 4

HALF TRACK USE: AN EXAMPLE

1XXXXXXXXXXX

- head group number ($0\text{-}7_8$)
- "1" if odd sectors, "0" if even sectors
- track number ($0\text{-}177_8$)

Sector numbers maintained by the system (such as the Current Sector in an FST entry) are logical sector numbers, and refer to a sector within a half track. In the outer zones, sectors within a half track are numbered $0\text{-}77_8$: In the inner zones, sectors within a half track are numbered $0\text{-}61_8$. To convert a logical sector number to a physical sector number, the system shifts the logical sector number left one place and inserts the $2^4$ bit from the half track address into the low-order bit position. For example, consider logical sector $77_8$ ($63_{10}$) in a half track composed of the odd-numbered sectors in a physical track. In this case, the $2^4$ bit of the half track address will be a "1". By shifting the logical sector left one place and inserting the "1" bit from the $2^4$ bit position of the half track address, we obtain $177_8$ ($127_{10}$) for the physical sector number. For the remainder of our discussion, a reference to "sector number" will refer to the logical sector number unless otherwise described.

For files recorded on the disk, the physical record is, of course, the sector. A logical record may be composed of several sectors. The format of the physical record is shown in figure 5. $502_8$ bytes are always written in each sector. The first two bytes written are control bytes: the remaining $500_8$ bytes are data bytes. Control byte 2 contains the number of useful central memory words in this sector: If control byte 2 contains $100_8$, all $500_8$ bytes in this sector contain useful information. A sector in which control byte 2 contains less than $100_8$ is called a short sector, and is interpreted as a record mark. A logical record may comprise several full sectors, but is always terminated by a short sector. If the data to be recorded as a logical record is a multiple of $100_8$ CM words, the system will write, as the record mark, a sector in which control byte 2 contains zero.

Control byte one points to the next physical record in this file. If the next sector is on the same half track, then this byte contains the

| CONTROL BYTE 1 | CONTROL BYTE 2 | 320 BYTES ALWAYS | WRITTEN |
|---|---|---|---|

CONTROL BYTE 2 → NUMBER OF USEFUL CM WORDS IN THIS SECTOR

CONTROL BYTE 1 → POINTER TO NEXT SECTOR

- SECTOR NUMBER ($0 - 77_8$) IF ON SAME HALF TRACK
- HALF TRACK NUMBER IF ON ANOTHER HALF TRACK

| CONTROL BYTE 1 | CONTROL BYTE 2 | RECORD |
|---|---|---|
| NON-ZERO | $100_8$ | "FULL" SECTOR: PART OF A LOGICAL RECORD |
| NON-ZERO | NON-ZERO, $<100_8$ | "SHORT" SECTOR: PART OF A LOGICAL RECORD; RECORD MARK |
| NON-ZERO | ZERO | "SHORT" SECTOR: RECORD MARK |
| ZERO | ZERO | FILE MARK |

Figure 5

DISK FILE PHYSICAL RECORD FORMAT

number of that sector. If the next sector is on another half track, then this byte contains the half track address for that half track. (The file would be continued beginning with sector zero of the new half track.)

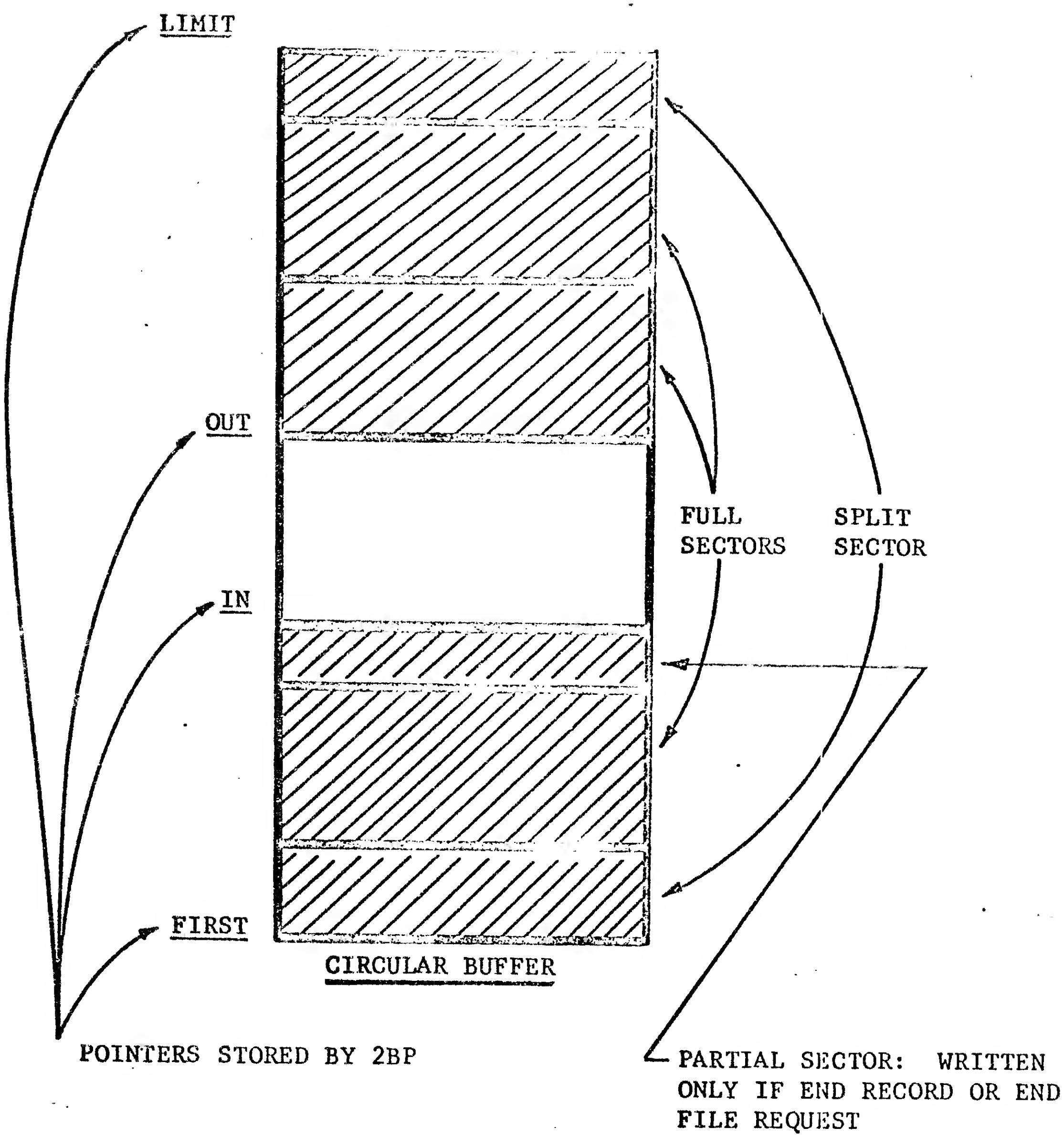
At the end of each write operation, the system writes a file mark. The Current Sector byte of the FST entry is not incremented to reflect this file mark sector, so the effect is equivalent to writing a file mark and backspacing over it. On the disk, a file mark is a sector in which both control bytes contain zero.

The Disk Write Overlay, 2WD

Disk write requests by users are executed by CIO's overlay 2WD. This overlay is also used by 1LJ and 1LT in loading jobs on the disk. Before calling 2WD, CIO calls the 2BP overlay to check the legality of the buffer parameters FIRST, IN, OUT, and LIMIT. After checking these parameters, 2BP searches the File Name Table for the file name specified in the CIO call (i.e., in the first word of the argument list). When found, 2BP stores the address of the corresponding FST entry. Should the file name not be found in the FNT, 2BP constructs an FNT entry for this file. Finally, 2BP clears the $2^0$ bit in the buffer status byte of the FST entry to reserve the file.

CIO then calls 2WD. (Refer to the flow chart on page A-1.) 2WD reads the FST entry for the file and extracts the equipment number from byte one. The equipment number is added to the EST base address, and the EST entry read. The channel number from byte 2 of the EST entry is then inserted in the appropriate I/O instructions.

The output data in the circular buffer may appear as a contiguous block, or may wrap around the buffer, as illustrated in figure 6. In computing the total number of sectors in the circular buffer, then, the 2WD routine first subtracts OUT from IN. If the difference is positive, then this difference is the total number of words to be written, and 2WD shifts off the lower six bits of this word count in order to obtain the equivalent number of sectors. If OUT-IN is negative, the value of LIMIT is added to the difference and FIRST subtracted to obtain the




## BUFFER PARAMETER PROCESSING

1. COMPUTE TOTAL NUMBER OF WORDS IN OUTPUT AREA

2. COMPUTE TOTAL NUMBER OF SECTORS IN OUTPUT AREA BY SHIFTING TOTAL WORD COUNT RIGHT 6 PLACES

3. COMPUTE NUMBER OF WORDS BETWEEN OUT AND LIMIT

4. COMPUTE NUMBER OF SECTORS BETWEEN OUT AND LIMIT BY SHIFTING OUT-LIMIT WORD COUNT RIGHT 6 PLACES

5. EXTRACT LOW-ORDER 6 BITS OF OUT-LIMIT WORD COUNT: THIS GIVES THE NUMBER OF WORDS IN THAT PART OF THE SPLIT SECTOR BETWEEN OUT AND LIMIT

6. SUBTRACT NUMBER OF WORDS COMPUTED IN (5) FROM $100_8$ TO GET NUMBER OF WORDS TO BE READ FROM THE BUFFER BEGINNING AT FIRST IN ORDER TO COMPLETE THE SPLIT SECTOR

7. SET UP INSTRUCTIONS FOR PROCESSING THE SPLIT SECTOR

CIRCULAR BUFFER PARAMETER PROCESSING - 2WD OVERLAY

Figure 6

total word count and, from that, the equivalent number of sectors.

Regardless of whether the data is contiguous or wraps around the buffer, 2WD proceeds on the assumption that the data does wrap around, and proceeds to compute the values needed to process the wraparound case. The steps involved are listed in figure 6. These values, although always computed, are not required in the contiguous case: in either case, the terminal path is entered when the total sector count is reduced to zero. By computing these values regardless of whether the data is contiguous in the buffer or wraps around the buffer, computations during the period when the disk is actively in use are reduced.

Next, 2WD picks up the channel number from the EST entry and requests reservation of that channel from MTR. The Current Track byte of the FST entry for this file is then examined. If this byte is zero, then this file has not previously been used. A half track assignment is requested from MTR: MTR returns a half track address to the requestor in byte one of the first word in the message buffer. If no half track is available, MTR will return a zero byte to the requestor: 2WD then inserts an error message in the dayfile and aborts the control point after dropping the channel reservation. 2WD now has the address of the half track where the next operation is to be performed, and proceeds to position the disk to this half track. This half track address is compared with byte 2 of the TRT pointer word for this disk, and repositioning or head group selection performed only if required. Byte 2 of the TRT pointer is then updated.

2WD next requests another half track assignment from MTR. This half track is a spare: by keeping it available, it is possible for 2WD to switch head groups within the group switch gap if this action should be required when the end of the current half track is reached.

The transfer of data from the buffer to the disk then begins. 2WD reads $100_8$ words from central memory into peripheral processor memory, sets control bytes one and two, and then writes the completed sector to the disk. As each sector is written, the number of the sector is examined to determine if the end of the half track is reached. To do this, 2WD compares the sector number with byte 4 of the TRT pointer word (if head

group number = 0-3) or byte 5 of the TRT pointer word (if head group number = 4-7). These bytes contain the values $100_8$ and $62_8$, respectively.

If the end of the half track has been reached, 2WD positions the disk to the spare half track: again, the half track address is compared with byte 2 of the TRT pointer word and positioning or head group selection performed only if required. After initiating any repositioning which might be required, 2WD requests a spare half track from MTR.

2WD continues reading $100_8$-word blocks from central memory and writing them to the disk until it recognizes that there is not enough data in the circular buffer for a complete sector. (Some part of a sector may still, however, remain.) 2WD then examines the buffer status contained in byte 5 of the FST entry to see if an end record was requested ($2^4$ bit = 1). If an end record was requested, 2WD writes a short sector to the disk. If any data remained in the circular buffer, it will be written in this short sector: otherwise, control byte 2 will simply be set to zero.

After the last data sector has been written to the disk, 2WD writes a file mark - a sector with both control bytes equal to zero. The Current Sector byte of the FST entry is not, however, incremented to reflect the writing of this file mark: the next write to this file will write over the file mark sector. After the file mark has been written, 2WD requests MTR to drop the spare half track assignment and to release the channel reservation.

If no end record function was requested, 2WD simply updates the OUT pointer before returning control to CIO: There may still be some data in the circular buffer. If an end record function was requested, no data remains in the buffer: 2WD therefore sets IN = OUT = FIRST to indicate that the buffer is empty.

When control is returned to CIO, CIO sets the $2^0$ bit of the buffer status in the FST entry to 1 to indicate that the file is no longer in use, and sets the $2^0$ bit of the buffer status in the calling program's argument list to 1 to indicate to the calling program that the operation has been completed.

The Disk Read Overlay, 2RD

Disk read requests by users are executed by CIO's overlay 2RD. This overlay is also used by 1DJ and 1TD. The processing performed by 2BP in this case is identical to that performed in the case of 2WD. On entry, 2RD reads the FST entry for the file, picks up the equipment number from byte one, and uses this number to obtain the EST entry. The channel number from the EST entry is then set in the I/O instructions.

2RD then proceeds to compute the number of sectors which can be loaded into the circular buffer. If there is not room for a full sector, control is returned to CIO. The data to be read may fit in the buffer in a contiguous block, or may wrap around the buffer. The computation of the values (total word count, total sector count, etc.) used in controlling the transfer of data to the buffer is performed in a manner similar to 2WD. Again, the wraparound case is assumed.

The Current Track byte of the FST entry is examined. If this byte is zero, the file has not been used before and so contains no data. 2RD sets the buffer status to indicate a file mark and returns control to CIO.

2RD requests a channel reservation from MTR and positions the disk to the half track address contained in the FST entry's Current Track byte. As in all disk routines, the half track address is compared with the disk position specified in the TRT pointer, and repositioning or head group switching performed only if necessary.

2RD then uses the Current Sector byte of the FST entry to construct the read function code, and reads the specified sector into peripheral processor memory. A status request is then issued, and the response is examined to determine if a parity error occurred. In the event of a parity error, the system rereads the sector three times; once using the normal sampling method and twice at varied sampling margins. If the parity error re-occurs in each of the rereads, 2RD inserts an error message in the dayfile and stops (via a UJN 0 instruction). Since the halt occurs without the disk channel being released, all system activity will shortly cease (if this disk is the

system disk, disk 0). A dead start load will be necessary to reinitiate processing.

If the read was successful, 2RD examines the high-order six bits of control byte one: if these bits are zero, then this control byte contains a sector number, while if these bits are non-zero, this control byte contains a half track number. In the latter case, 2RD positions the disk to the new half track address. While any repositioning or head group switching which might be required is in process, 2RD transfers the number of words specified in control byte 2 from peripheral processor memory to the circular buffer, and updates the values used in controlling the transfer. If the sector just read was a full sector ($100_8$ CM words of data), and if there is enough room in the circular buffer for another full sector, 2RD loops to read the next sector from the disk.

If the last sector read was a short sector, then the end of a logical record has been reached, and the buffer status is set to reflect a record mark. If the end of logical record has been reached, or if there is not enough room in the circular buffer for a full sector, 2RD requests MTR to release the channel reservation, updates the IN pointer in the calling program's argument list, and returns control to CIO. CIO updates the buffer status in the FST entry to release the file reservation, and updates the buffer status in the calling program's argument list to indicate that the operation has been completed.

If, after reading the last logical record in a file, the calling program issues another read to the file, the file mark will be read. The processing proceeds as described above: 2RD reads a sector whose address is specified in the Current Track and Current Sector bytes of the FST entry. Since control byte 2 is zero, 2RD recognizes this as a short sector, sets the buffer status to reflect a record mark, and releases the channel. 2RD then examines control byte one; since this contains zero, the file mark is recognized and the buffer status set accordingly before returning control to CIO.

## The Backspace Disk Overlay, 2BD

Disk backspacing may take the form of a BCD backspace or, more commonly, a binary backspace. In either case, it is desired to backspace over a logical record, and it is assumed that any backspacing over logical records in the buffer has been done by the calling program. Backspacing over the physical records which may constitute a logical record is essentially a matter of backspacing over two sectors and then reading a sector.

2BD uses a subroutine to backspace over a sector. (See flow chart on page A-5.) This subroutine examines the Current Sector byte of the FST entry, and, if non-zero, subtracts one from this number and exits. This is equivalent to backspacing over one physical record (i.e., one sector). If the Current Sector number is zero, then the preceding physical record is on another half track. In this case, the subroutine stores the Current Track byte from the EST entry for this file, since it will have to search the file for a sector which has this half track address contained in control byte one.

The subroutine rewinds the file by picking up the Beginning Track byte from the FST entry. (Should the Beginning Track byte be equal to the Current Track byte, the subroutine exits, since this indicates that the system has backspaced over all physical records in this file.) After rewinding the file, the subroutine reads each sector in the file until it finds a sector with the desired half track address in control byte one. The number of this sector is then stored, and control returned to the calling routine. A backspace operation on a file of any size may take considerable time if it should become necessary to rewind the file and search forward.

A binary backspace on the disk consists of backspacing over two sectors (using the subroutine described above) and reading a sector until a short record is found, indicating the end of a logical record. 2BD sets the circular buffer pointers IN and OUT equal to FIRST, and returns control to CIO. CIO updates the buffer status in the FST entry and in the calling program's argument list before exiting.

It is also possible to issue a BCD backspace to the disk. For the disk, as for 1" tape (but not for 1" tape), a logical BCD record consists of a series of central memory words presumably containing display code data, terminated by a central memory whose low-order byte (byte 5) is zero.

The BCD backspace begins with the computation of the amount of data left in the buffer as a result of the last read. This quantity, referred to as D, is equal in IN-OUT if the data in the buffer is contiguous, or IN-OUT + LIMIT-FIRST if the data wraps around the buffer. This data was left in the buffer as a result of the last read, and may have been stored on the disk in several sectors. The system assumes that the calling program will backspace within the buffer, and so, before beginning a logical BCD record backspace on the disk, 2BD will backspace the disk a number of sectors equivalent to the amount of data contained in the buffer. This quantity is represented by D.

2BD therefore backspaces over a sector (by the same subroutine used in binary backspacing and described earlier) and reads that sector into peripheral processor memory. The sector length in control byte 2 is then compared with D: if less than D, then this sector is assumed to contain data which has already been read into the buffer. 2BD then decreases D by this amount, backspaces over this sector and the sector preceding it, and then reads a sector. The process of backspacing, reading, and reducing D is repeated until a sector is read whose length is greater than the present value of D: this sector could not entirely be part of the read data in the buffer, and so must be searched for a logical record. 2BD transfers this sector from peripheral processor memory to the circular buffer beginning at FIRST. If D is still non-zero, then part of this sector contains data residing in the buffer at the time the backspace was requested, and presumably has been searched by the calling program: 2BD therefore sets the OUT pointer to FIRST + sector length - D. At the same time, the IN pointer is set to reflect the transfer of the sector to the buffer.

2BD then searches each word in the buffer from OUT - 1 down to FIRST until a word with a zero low-order byte is found, indicating the end of a logical BCD record. When the end of the record is found, 2BD updates the IN and OUT pointers in the calling program's argument list, and

returns control to CIO. OUT now points to the first word following the end of the logical record. If no zero low-order byte was found, then 2BD backspaces two sectors and reads one, and then repeats the buffer search.

The Drop Track Overlay, 2DT

When CIO receives a disk write request, it first calls the 2BP overlay to check the legality of the buffer parameters and to search the FNT for the file name. CIO then reads the EST entry for this file, and examines the buffer status in byte 5. If the buffer status indicates that the last operation performed on this file was a read operation, then an overlay, 2DT, is called to drop the subsequent portion of the file. In effect, then, if some part of a file is read and it is then decided to write to that file, the remainder of the file is erased.

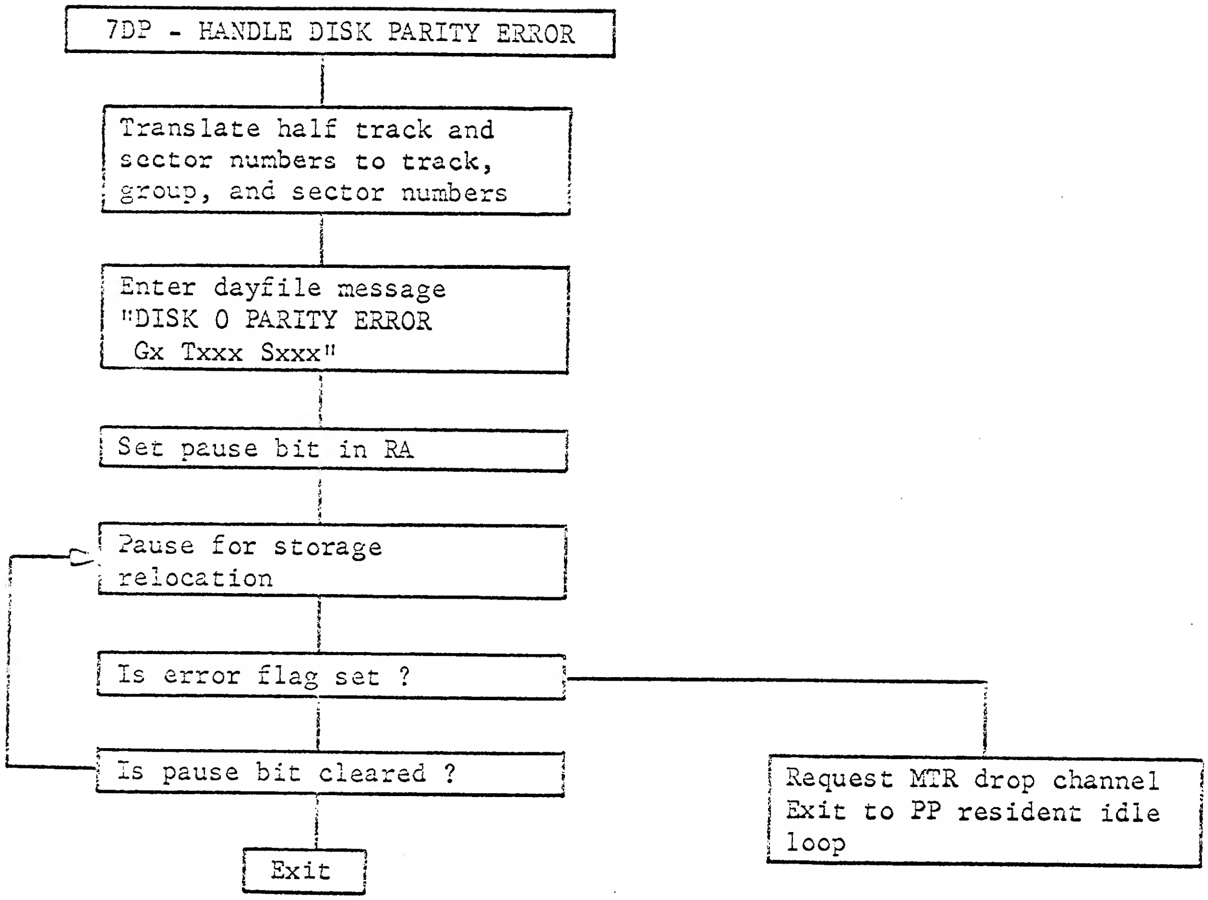
The flow chart for the 2DT overlay is shown on page A-3 of the attached flow charts. The routine picks up the Current Track byte and Current Sector byte from the FST entry for the file, and reads the sector at this address. If this sector is a file mark, 2DT returns control to CIO. If control byte one of this sector contains a half track address, 2DT requests MTR to drop this half track reservation. MTR then clears the bit in the Track Reservation Table corresponding to this half track address. 2DT positions the disk to this half track address and begins reading sectors until a file mark is found or the end of the half track is reached. The process of reading and dropping half tracks continues until the end of the file is reached.

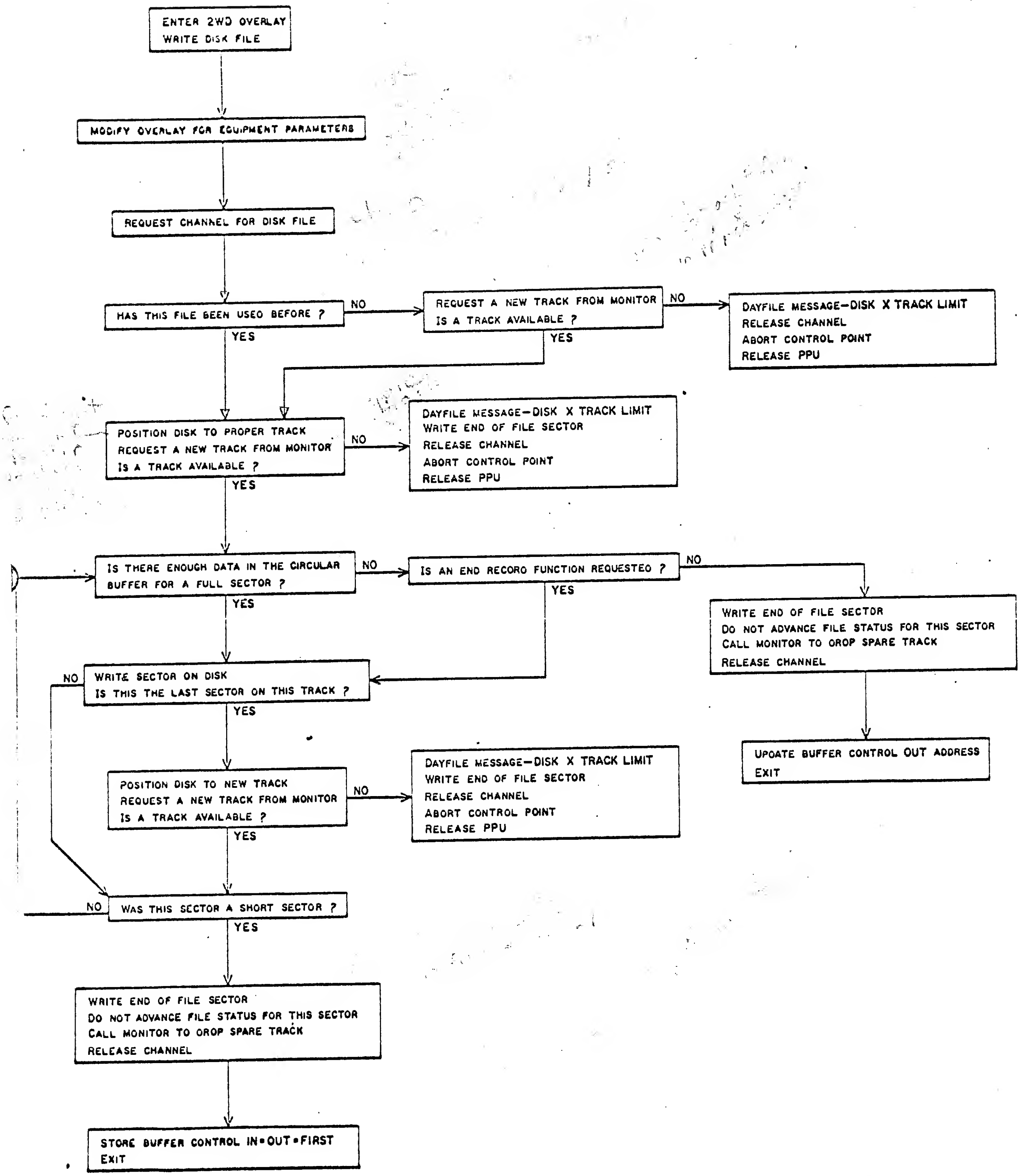
At the end of a job, all local files associated with the job are dropped. For disk files, a process similar to that described above is required to release half track reservations. This is performed for 1AJ by the 2DF overlay. 2DF differs from 2DT in that 2DF drops files assigned to other equipment as well as those assigned to the disk, and 2DF drops all the half tracks reserved by a file, not just those following the half track specified in the Current Track byte of the FST entry. 2DF is also called by 1DJ and 1TD when printing files or writing files on tape.

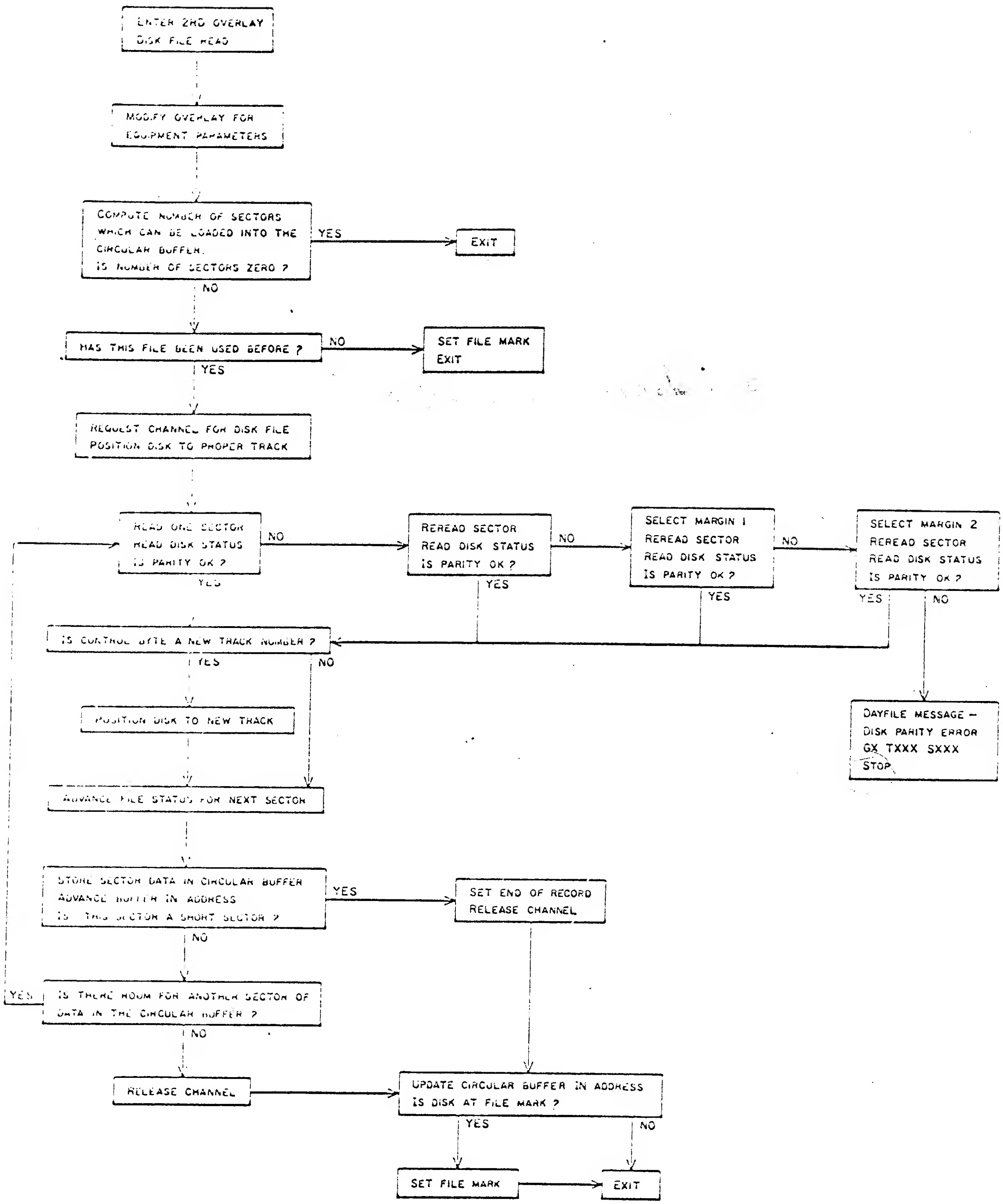
DISK PARITY ERROR PROCESSING

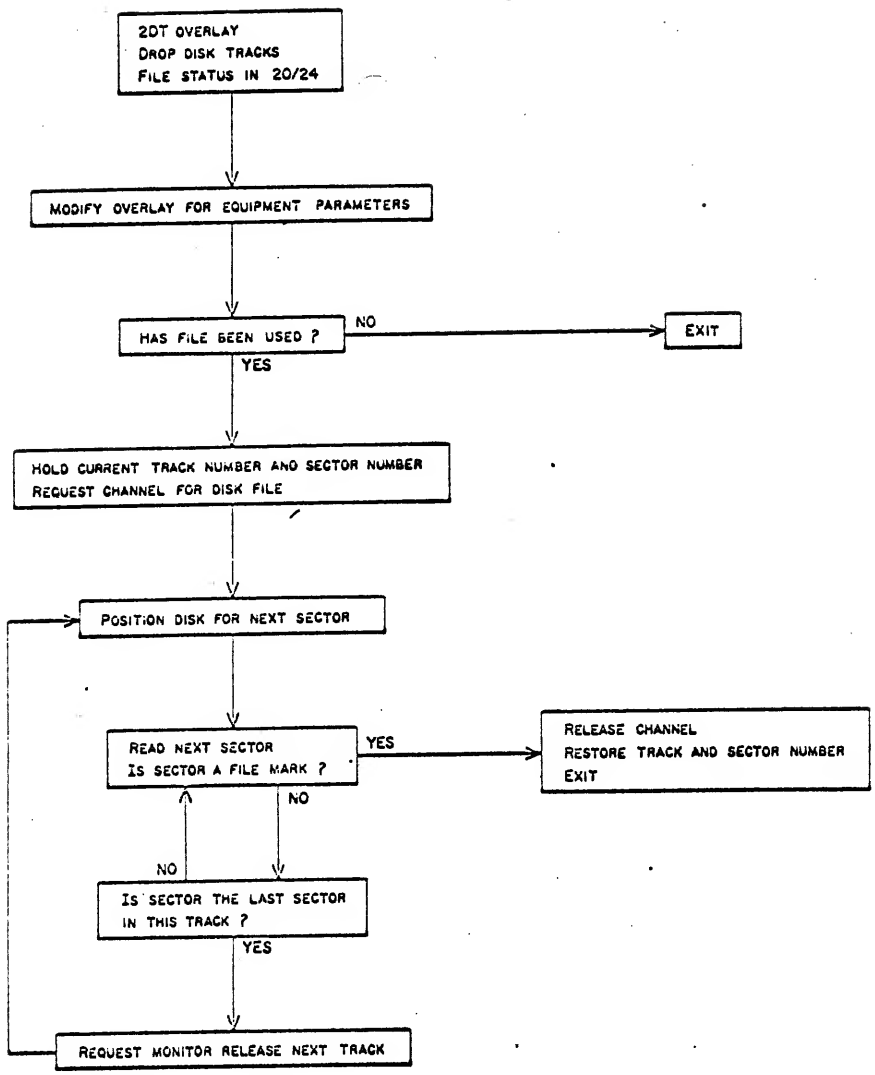
When a disk parity error persists after three re-reads in any of the disk read routines, control is transferred to a peripheral processor resident routine tagged DPAR at location 0200. This routine loads the 7DP overlay - Disk Parity Error Handler - into peripheral processor memory at location 7553. Note that this overlay is loaded above the memory area used to buffer the sector from the disk: the latter occuppies locations 7000 - 7501.

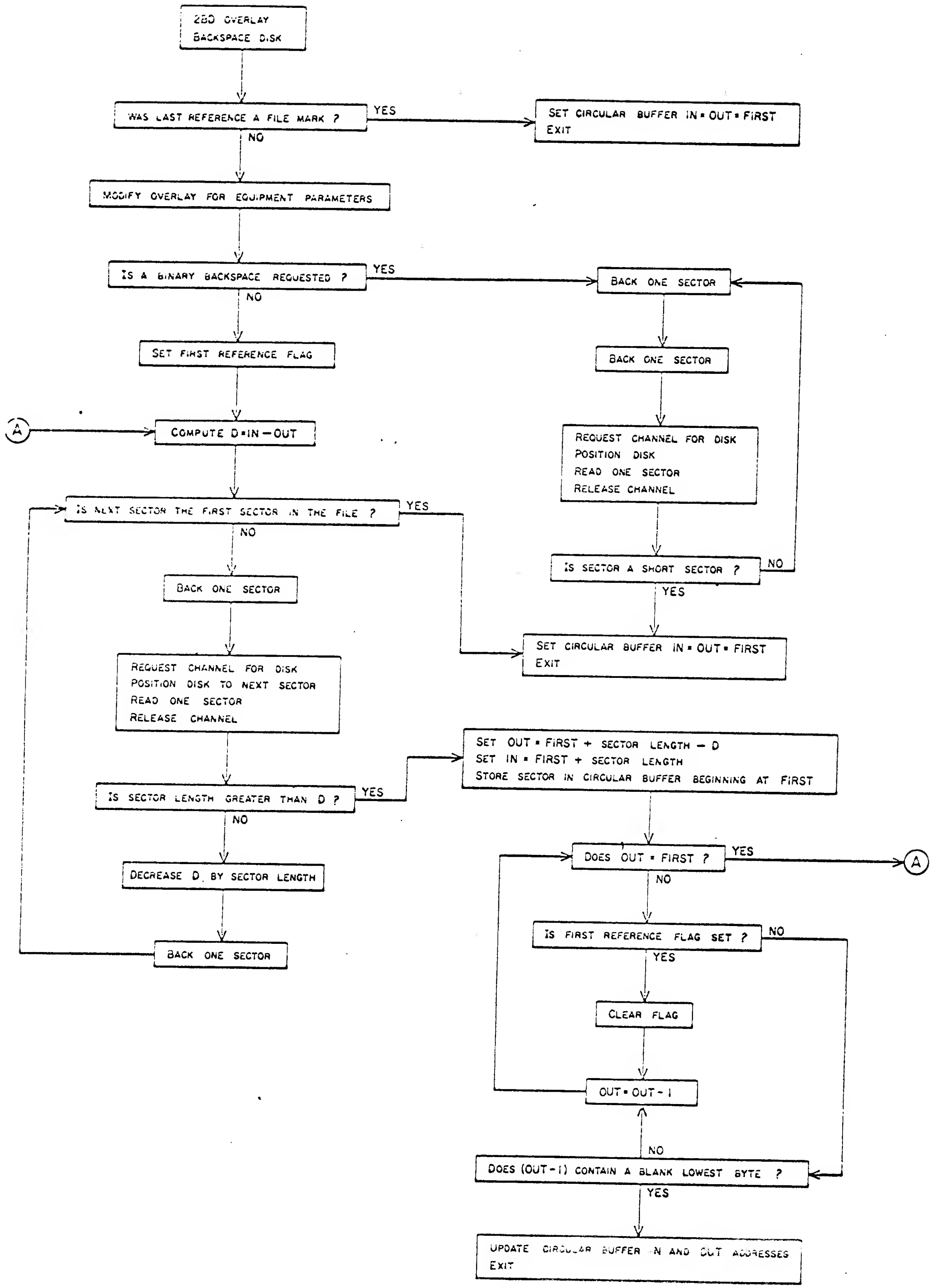
On entry to 7DP, the logical sector number and the half track number are converted to the physical sector number, head group number, and track number. The error message "DISK 0 PARITY ERROR Gx Txxx Sxxx" is entered in the dayfile. Next, the pause bit is set in location RA ($2^0$ bit of byte 4). 7DP then enters a loop in which it pauses for storage relocation, tests the error flag in byte 1 of word 20 in the control point area, and tests the pause bit in RA. This loop is repeated until the operator types the DSD keyboard entry n.GO (which clears the pause bit in RA) or the entry n.DROP (which sets an error flag). If the error flag is set, 7DP requests MTR to drop the channel, and exits to the resident idle loop. If the operator clears the pause bit, execution proceeds as if no error had occurred.

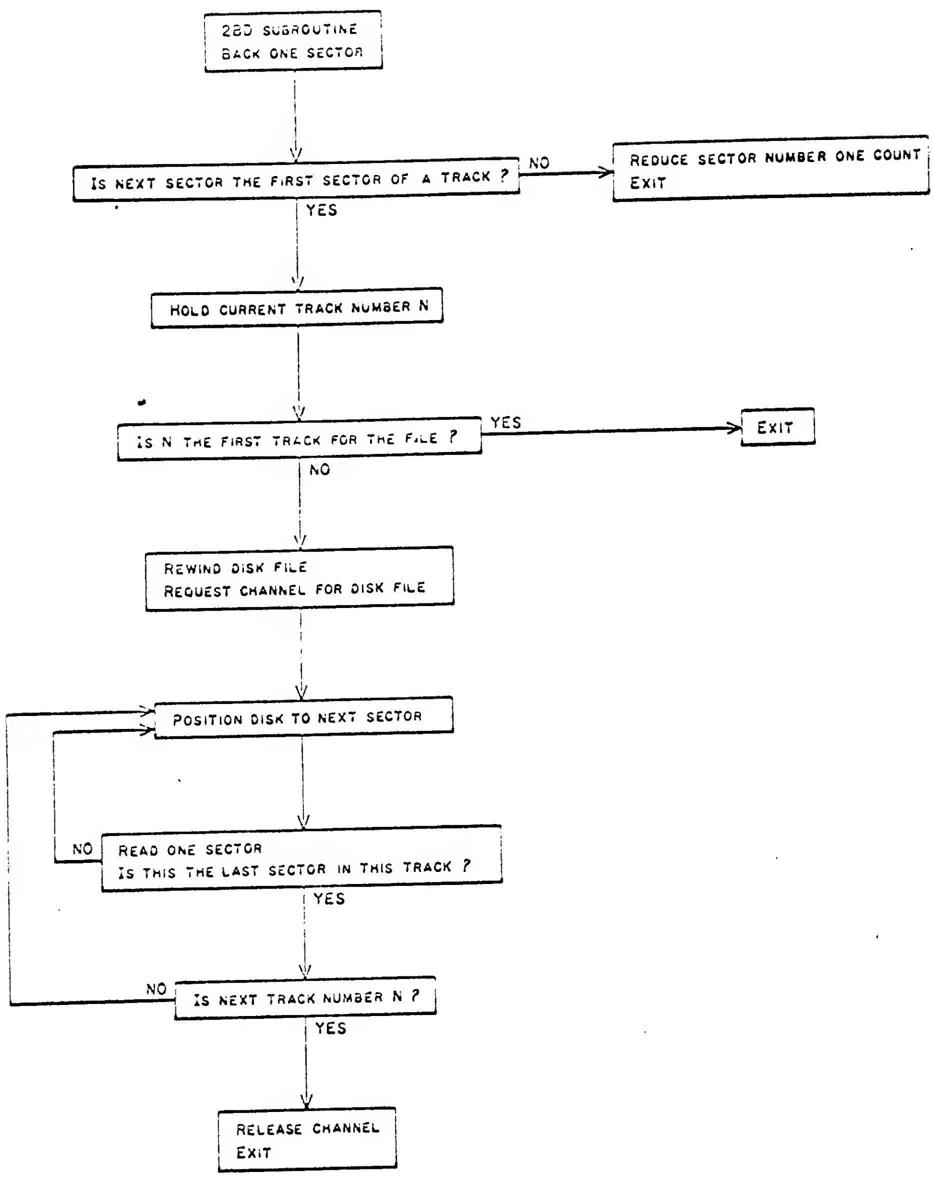

7DP - HANDLE DISK PARITY ERROR
Translate half track and sector numbers to track, group, and sector numbers
Enter dayfile message "DISK 0 PARITY ERROR Gx Txxx Sxxx"
Set pause bit in RA
Pause for storage relocation
Is error flag set ?
Is pause bit cleared ?
Exit
Request MTR drop channel Exit to PP resident idle loop


ENTER 2WD OVERLAY
WRITE DISK FILE
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
REQUEST CHANNEL FOR DISK FILE
HAS THIS FILE BEEN USED BEFORE ?
NO
REQUEST A NEW TRACK FROM MONITOR
IS A TRACK AVAILABLE ?
NO
DAYFILE MESSAGE—DISK X TRACK LIMIT
RELEASE CHANNEL
ABORT CONTROL POINT
RELEASE PPU
YES
YES
POSITION DISK TO PROPER TRACK
REQUEST A NEW TRACK FROM MONITOR
IS A TRACK AVAILABLE ?
NO
DAYFILE MESSAGE—DISK X TRACK LIMIT
WRITE END OF FILE SECTOR
RELEASE CHANNEL
ABORT CONTROL POINT
RELEASE PPU
YES
IS THERE ENOUGH DATA IN THE CIRCULAR
BUFFER FOR A FULL SECTOR ?
NO
IS AN END RECORD FUNCTION REQUESTED ?
NO
YES
WRITE END OF FILE SECTOR
DO NOT ADVANCE FILE STATUS FOR THIS SECTOR
CALL MONITOR TO DROP SPARE TRACK
RELEASE CHANNEL
UPDATE BUFFER CONTROL OUT ADDRESS
EXIT
YES
NO
WRITE SECTOR ON DISK
IS THIS THE LAST SECTOR ON THIS TRACK ?
YES
POSITION DISK TO NEW TRACK
REQUEST A NEW TRACK FROM MONITOR
IS A TRACK AVAILABLE ?
NO
DAYFILE MESSAGE—DISK X TRACK LIMIT
WRITE END OF FILE SECTOR
RELEASE CHANNEL
ABORT CONTROL POINT
RELEASE PPU
YES
NO
WAS THIS SECTOR A SHORT SECTOR ?
YES
WRITE END OF FILE SECTOR
DO NOT ADVANCE FILE STATUS FOR THIS SECTOR
CALL MONITOR TO DROP SPARE TRACK
RELEASE CHANNEL
STORE BUFFER CONTROL IN•OUT•FIRST
EXIT


ENTER 2ND OVERLAY
DISK FILE READ
MODIFY OVERLAY FOR
EQUIPMENT PARAMETERS
COMPUTE NUMBER OF SECTORS
WHICH CAN BE LOADED INTO THE
CIRCULAR BUFFER.
IS NUMBER OF SECTORS ZERO ?
YES
EXIT
NO
HAS THIS FILE BEEN USED BEFORE ?
NO
SET FILE MARK
EXIT
YES
REQUEST CHANNEL FOR DISK FILE
POSITION DISK TO PROPER TRACK
READ ONE SECTOR
READ DISK STATUS
IS PARITY OK ?
NO
REREAD SECTOR
READ DISK STATUS
IS PARITY OK ?
NO
SELECT MARGIN 1
REREAD SECTOR
READ DISK STATUS
IS PARITY OK ?
NO
SELECT MARGIN 2
REREAD SECTOR
READ DISK STATUS
IS PARITY OK ?
YES
YES
YES
YES
NO
IS CONTROL BYTE A NEW TRACK NUMBER ?
YES
NO
POSITION DISK TO NEW TRACK
DAYFILE MESSAGE –
DISK PARITY ERROR
GX TXXX SXXX
STOP
ADVANCE FILE STATUS FOR NEXT SECTOR
STORE SECTOR DATA IN CIRCULAR BUFFER
ADVANCE BUFFER IN ADDRESS
IS THIS SECTOR A SHORT SECTOR ?
YES
SET END OF RECORD
RELEASE CHANNEL
NO
YES
IS THERE ROOM FOR ANOTHER SECTOR OF
DATA IN THE CIRCULAR BUFFER ?
NO
RELEASE CHANNEL
UPDATE CIRCULAR BUFFER IN ADDRESS
IS DISK AT FILE MARK ?
YES
NO
SET FILE MARK
EXIT


2DT OVERLAY
DROP DISK TRACKS
FILE STATUS IN 20/24
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
HAS FILE BEEN USED ?
NO
EXIT
YES
HOLD CURRENT TRACK NUMBER AND SECTOR NUMBER
REQUEST CHANNEL FOR DISK FILE
POSITION DISK FOR NEXT SECTOR
READ NEXT SECTOR
IS SECTOR A FILE MARK ?
YES
RELEASE CHANNEL
RESTORE TRACK AND SECTOR NUMBER
EXIT
NO
NO
IS SECTOR THE LAST SECTOR
IN THIS TRACK ?
YES
REQUEST MONITOR RELEASE NEXT TRACK


Z80 OVERLAY
BACKSPACE DISK
WAS LAST REFERENCE A FILE MARK ?
YES
NO
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT
MODIFY OVERLAY FOR EQUIPMENT PARAMETERS
IS A BINARY BACKSPACE REQUESTED ?
YES
NO
BACK ONE SECTOR
BACK ONE SECTOR
SET FIRST REFERENCE FLAG
A
COMPUTE D = IN − OUT
REQUEST CHANNEL FOR DISK
POSITION DISK
READ ONE SECTOR
RELEASE CHANNEL
IS NEXT SECTOR THE FIRST SECTOR IN THE FILE ?
YES
NO
IS SECTOR A SHORT SECTOR ?
NO
YES
BACK ONE SECTOR
SET CIRCULAR BUFFER IN = OUT = FIRST
EXIT
REQUEST CHANNEL FOR DISK
POSITION DISK TO NEXT SECTOR
READ ONE SECTOR
RELEASE CHANNEL
SET OUT = FIRST + SECTOR LENGTH − D
SET IN = FIRST + SECTOR LENGTH
STORE SECTOR IN CIRCULAR BUFFER BEGINNING AT FIRST
IS SECTOR LENGTH GREATER THAN D ?
YES
NO
DOES OUT = FIRST ?
YES
A
NO
DECREASE D BY SECTOR LENGTH
IS FIRST REFERENCE FLAG SET ?
NO
YES
BACK ONE SECTOR
CLEAR FLAG
OUT = OUT − 1
NO
DOES (OUT − 1) CONTAIN A BLANK LOWEST BYTE ?
YES
UPDATE CIRCULAR BUFFER IN AND OUT ADDRESSES
EXIT


Z8D SUBROUTINE
BACK ONE SECTOR
IS NEXT SECTOR THE FIRST SECTOR OF A TRACK ?
NO
REDUCE SECTOR NUMBER ONE COUNT
EXIT
YES
HOLD CURRENT TRACK NUMBER N
IS N THE FIRST TRACK FOR THE FILE ?
YES
EXIT
NO
REWIND DISK FILE
REQUEST CHANNEL FOR DISK FILE
POSITION DISK TO NEXT SECTOR
NO
READ ONE SECTOR
IS THIS THE LAST SECTOR IN THIS TRACK ?
YES
NO
IS NEXT TRACK NUMBER N ?
YES
RELEASE CHANNEL
EXIT

CONTROL DATA CORPORATION

Product Marketing Management

# SYSTEM/OPERATOR COMMUNICATION

Chippewa Operating System

May 2, 1966

# SYSTEM/OPERATOR COMMUNICATION

## CONSOLE AND DISPLAY SCOPES

### SYSTEM DISPLAY

SYSTEM DISPLAY CODES

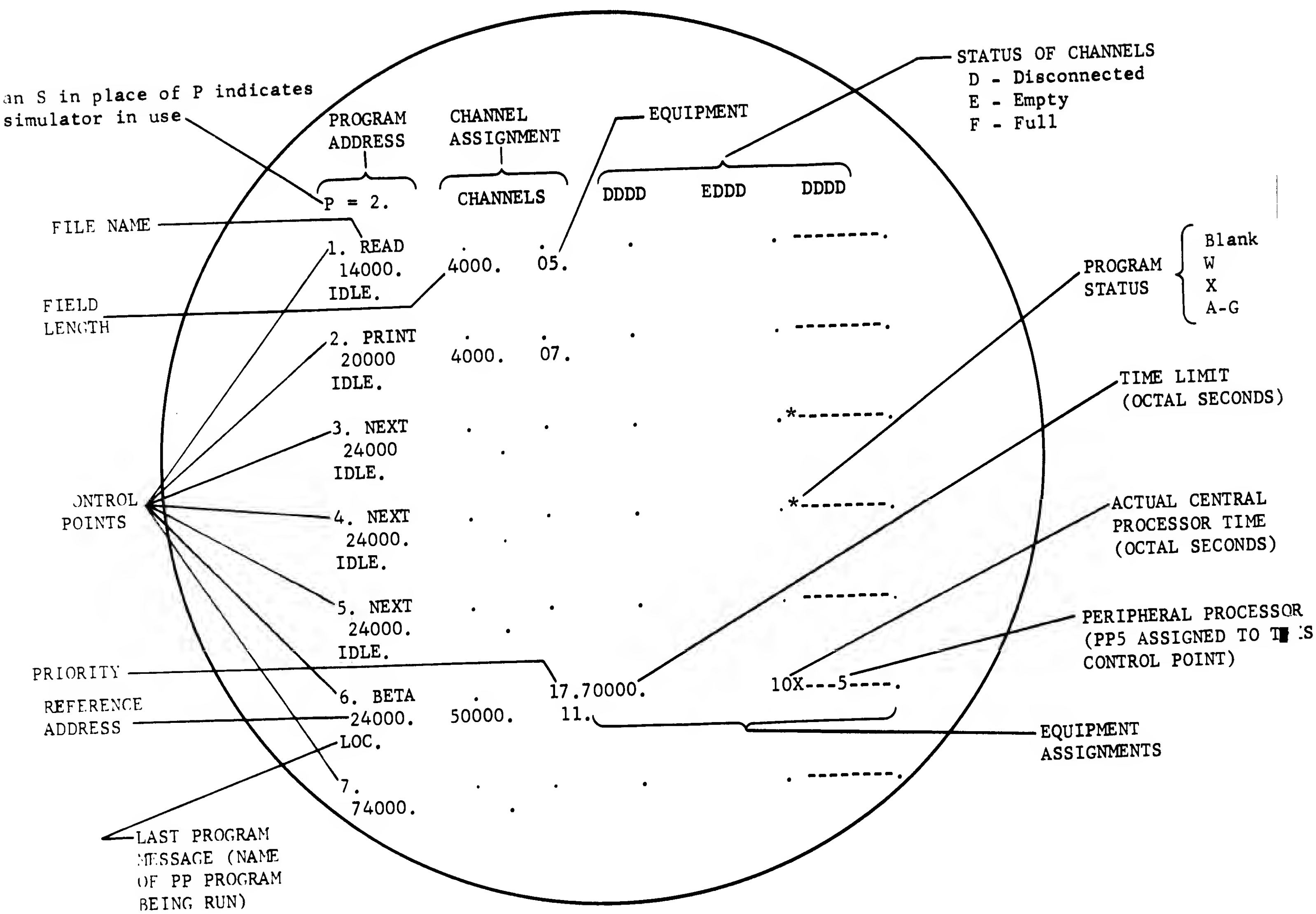
| Codes | Display |
|---|---|
| A | Dayfile |
| B | Job Status |
| C | Data Storage |
| D | Data Storage |
| E | Data Storage |
| F | Program Storage |
| G | Program Storage |
| H | Job Backlog |

JOB DISPLAY CODES

| Codes | | |
|---|---|---|
| A | Dayfile | |
| B | Job Status | |
| C | Data Storage | 5 groups of 4 octal digits per group |
| D | Data Storage | |
| E | Data Storage | |
| F | Program Storage | 4 groups of 5 octal digits per group |
| G | Program Storage | |

## SYSTEM DISPLAY

### JOB STATUS (B) DISPLAY



## DAYFILE (A) DISPLAY

ACTUAL TIME

Hours

Min

Sec.

Name of JOB to which message belongs

SYSTEM TAPE LABEL

JOB NAME

```
00.12.55.   PROGRAMMING CHECKOUT

00.00.16.  MERGE  .  MERGE,7,1000,1000.
00.00.17.  MERGE  .  ASSIGN 50,A.
00.00.17.  MERGE  .    (50 ASSIGNED)
00.00.17.  MERGE  .  ASSIGN 51,B.
00.00.17.  MERGE  .    (51 ASSIGNED)
00.00.18.  MERGE  .  REWIND (F)
00.00.25.  MERGE  .  REWIND (F)
00.00.25.  MERGE  .  COPYBF (F.D)
00.00.27.  BETA   .  READ.
00.00.30.  MERGE  .  REWIND (F)
00.00.30.  MERGE  .  REWIND (F)
00.00.30.  MERGE  .  CP 006.(D) SEC.
00.00.30.  MERGE  .  PP 019.421 SEC.
00.00.30.  MERGE  .  PRINT.
00.00.30.  MERGE  .  PP 000 SEC.
00.00.31.  BETA   .  PP 015 SEC.
00.00.31.  BETA   .  BETA,77,70000,50000.
00.00.31.  BETA   .  DIS.
00.01.04.  BETA   .  INPUT.
00.01.05.  BETA   .  LOC.
00.01.11.  BETA   .  BUFFER ARG ERROR.
00.01.12.  BETA   .  CP 002.575 SEC.
00.01.12.  BETA   .  PP 020.265 SEC.
00.01.12.  BETA   .  PRINT.
00.01.24.  BETA   .  PP 011 SEC.
00.04.05   BETA   .  READ
00.04.10   BETA   .  PP 015 SEC.
00.04.10   BETA   .  BETA,77,70000,50000.
00.04.10   BETA   .  DIS.
00.04.40   BETA   .  INPUT.
00.04.40   BETA   .  LOC.
```

This column represents the time each control statement was requested for execution.
(A total of 32 lines may be contained on the dayfile)

New dayfile information appears at the bottom of the screen automatically; old dayfile information is deleted at the top of the column as new times are entered into the dayfile.

This column represents the control statements introduced via card input and contains the system's history.

A summary of the day's total run may be printed out upon request.

NOTE: Dayfile display data will appear on the printout at the end of each job automatically

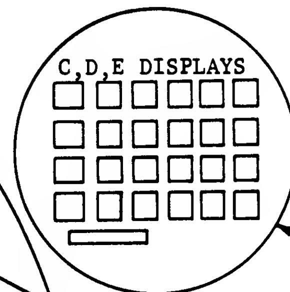
## Storage (C Thru H) Displays

00.17.21. PROGRAMMING CHECKOUT

ADDRESS — CONTENTS OF ADDRESS

| | | | | |
|---|---|---|---|---|
| 010000 | 32670 | 62330 | 60133 | 70520 |
| 010001 | 30645 | 26010 | 14316 | 53261 |
| 010002 | 07125 | 06232 | 00101 | 43165 |
| 010003 | 32610 | 70401 | 00235 | 00000 |
| 010004 | 20002 | 06602 | 00053 | 11413 |
| 010005 | 02000 | 76101 | 00010 | 00000 |
| 010006 | 00000 | 00000 | 00000 | 00000 |
| 010007 | 35022 | 40000 | 00000 | 00126 |
| 010010 | 01000 | 00002 | 00224 | 13024 |
| 010011 | 12020 | 51430 | 03100 | 70705 |
| 010012 | 02002 | 56130 | 17000 | 00200 |
| 010013 | 24010 | 32100 | 00020 | 02561 |
| 010014 | 02002 | 56102 | 00210 | 13004 |
| 010015 | 10660 | 56730 | 00346 | 23464 |
| 010016 | 30615 | 46334 | 05140 | 06010 |
| 010017 | 30625 | 41330 | 05341 | 43053 |
| 010020 | 10063 | 15510 | 06315 | 41602 |
| 010021 | 62103 | 06434 | 13306 | 53414 |
| 010022 | 30531 | 00631 | 55100 | 65154 |
| 010023 | 16036 | 21001 | 00200 | 00000 |
| 010024 | 00000 | 00000 | 00000 | 00100 |
| 010025 | 00003 | 03102 | 00074 | 13032 |
| 010026 | 76000 | 20023 | 01140 | 03403 |
| 010027 | 30725 | 40430 | 32162 | 07600 |
| 010030 | 74002 | 00050 | 00710 | 02700 |
| 010031 | 04113 | 40137 | 04150 | 55501 |
| 010032 | 04020 | 67303 | 03000 | 07500 |
| 010033 | 02002 | 30112 | 02040 | 50200 |
| 010034 | 23011 | 22004 | 12305 | 10200 |
| 010035 | 07513 | 62210 | 03352 | 10100 |
| 010036 | 21000 | 00036 | 03110 | 50407 |
| 010037 | 30321 | 63076 | 00010 | 02115 |

DISPLAY AREA FOR OPERATOR TYPEIN (Always on left screen)




FOUR GROUPS OF FIVE-OCTAL DIGITS--(OR FIVE GROUPS OF FOUR-OCTAL DIGITS)

H DISPLAY

| INPUT | OUTPUT |
|---|---|
| Job 1 | Job N |
| Job 2 | Job X |
| | Job Y |

Job Names (Input stack)

Jobs stacked for output (If no jobs are waiting for output none will appear)

## SYSTEM DISPLAY (DSD)

**KEYBOARD ENTRIES** The keyboard, during the display of the overall system status, is used to initiate and control equipment assignment and job progress. The following table describes the keyboard codes and formats.

SYSTEM DISPLAY KEYBOARD ENTRIES

| Typein | Action Initiated |
|---|---|
| AUTO. | Used after dead start from system tape to initiate automatic job processing with card input and printer output. |
| STEP. | Selects a step mode for the operating system monitor in PP0. Requests from other PP's are processed when the keyboard space bar is pressed. High speed operation may be resumed by entering a period and depressing the carriage return key. |
| SIM. | Replaces the central processor with a simulator in a PP (the 007 location program); normally used in machine check out. Normal operation can be resumed by repeating SIM. Successive entries toggle the system between a simulated and a real central processor. The letter preceding central processor address on system display denotes the mode.<br>P = xxxxxx real central processor<br>S = xxxxxx simulated central processor |
| OFFxx. | Indicates to the system that equipment number xx must not be used, for example, during maintenance. |
| ONxx. | Returns equipment xx to the pool of available equipment. |
| TIME. 12.10.03, March 12, 1965. | Example of how to inform the system of clock time and date. The system uses this time (updated every second) for dayfile messages. If a TIME command is not given, the time since dead start is used. |

SYSTEM DISPLAY KEYBOARD ENTRIES (cont'd.)

| Typein | Action Initiated |
|---|---|
| CONTROL POINT COMMANDS TO DSD | |

Each of these commands is preceded by a control point number and a period. The numbers in the following examples are for illustrating format only.

| | |
|---|---|
| 4. DROP. | Drop job at control point 4. |
| 5. DIS. | Assign DIS package to control point 5. If there is no other console, DSD may relinquish the main console with the ASSIGN statement. |
| 5. ASSIGN10. | Assign main console to control point 5. |
| 1. READ. | Bring READ package to control point 1 for loading jobs from cards. This entry is needed if READ (usually initiated by AUTO) drops after reading a faulty job card. |
| 2. PRINT. | Bring print job to control point 2; normally initiated by AUTO, but may be brought in separately. |
| 7. LOAD. | Bring Load package to control point 7 to load jobs from tape to double file mark. |
| 6. DUMP. | Bring Dump package to control point 6 to dump output files on tape. |

(LOAD and DUMP request assignment of a tape unit).

(READ, PRINT, LOAD, DUMP, and NEXT have no effect if the control point already has a job name).

| | |
|---|---|
| 7. NEXT. | Bring job NEXT to control point 7 to look for a real job (on disk) for the control point. |
| 6. GO. | Continue job at control point 6 if it has come to a pause (usually as a result of a FORTRAN pause statement, or if repeated tape transfers after parity failure is not effective). |
| 5. ASSIGN53. | Assign equipment 53 to control point 5; normally used after a request for tape has been displayed, but any equipment may be assigned in this way. |
| 4. ONSW2. | Set sense switch 2 for FORTRAN program at control point 4. Settings are preserved at RA and at a word in the control point area. |
| 4. OFFSW3. | Turn off sense switch 3 for control point 4. |
| DCN11. | Disconnect channel $11_8$. (used by maintenance engineer) |
| FCN10. | Enter a zero function on channel $10_8$. (used by maintenance engineer) |

SYSTEM DISPLAY KEYBOARD ENTRIES (CONT'D)

| Type In | Action Initiated |
|---|---|
| 3.DIS. | At any unassigned control point, typing in DIS will bring up the Job Display. |

The values at the control point will be:

| | |
|---|---|
| JOB NAME | DIS |
| TIME LIMIT | $600_8$ SECONDS |
| FIELD LENGTH | $60000_8$ CM LOCATIONS |
| PRIORITY | 17 |

2.ENDn — A print job may be aborted without affecting other print operations. The Dayfile will be printed for the terminated print job where n is the number of the print job (from Display) to be terminated.

6.EXPRESS. — This will reserve the control point (control points) for jobs of the following types:

1. JOB cards with no priority, time limit, or field length.
2. JOB cards with TIME LIMIT equal or less than $100_8$ and FIELD LENGTH equal or less than $40,000_8$.

4.COMMENT. comments.

The remarks following COMMENT are entered in the Dayfile (and printed in the job Dayfile at the end of the listing) and displayed at the control point.

**JOB DISPLAY — DIS** A job display (DIS), similar to DSD, is used for information more relevant to a single job. Using DIS, the B display can demonstrate the exchange jump area of the job; central memory addresses relative to the job's reference address are used for data and program displays.

DIS can be called either from a control card (DIS.) or by a command to DSD. The job display package (DIS) may be called at any time during the execution of job. This package stops further automatic advance of the job control cards. The display covers only data pertaining to the particular job. The keyboard is used to advance the job control cards and to provide any two of the following displays in the same manner as for the DSD display.

JOB DISPLAY CODES

| Codes | | |
|---|---|---|
| A | Dayfile | |
| B | Job Status | |
| C | Data Storage | 5 groups of 4 octal digits per group |
| D | Data Storage | |
| E | Data Storage | |
| F | Program Storage | 4 groups of 5 octal digits per group |
| G | Program Storage | |

**CHANGING PRIORITY**

All jobs are assigned priorities from the job control card (0-17); a zero priority causes a job to be ignored. The operator may change the priority of any job through the keyboard, the range is 0-$77_8$ with $77_8$ being highest priority.

The operator may proceed as follows:

1. Type in: ENPR, dd.

   dd = a two-digit octal priority number

2. Press the carriage return key. If a priority change is attempted during a run, the program will stop (normal stop) and the operator may type in RCP. to resume central processor operation. A priority change may be made only when the job is assigned to a control point.

ENTRIES FOR CHANGING JOB DISPLAY CONTENTS

| | |
|---|---|
| ENP,12345. | Set P = 12345. (next instruction address, in exchange jump area). |
| ENA3,665000. | Set A3=665000 in exchange jump area. |
| ENB2,44. | Set B2=44 in exchange jump area. |
| ENX5,2223 4000 0000 0000 0200. (Spacing is unimportant) | Set X5=22234000000000000200 in exchange package. |
| ENEM,7. | Set Exit Mode = 7 in exchange jump area. |
| ENFL,10000. | Set FL=10000 in exchange jump area. (storage moved if necessary). |
| ENTL,200. | Set central processor time limit = $200_8$ seconds. |
| ENPR,5. | Set job priority = 5. |
| DCP. | Drop central processor and display exchange jump area (in display B). When DIS is used, the exchange jump area is displayed in any case if the job does not have status A, B etc. |
| RCP. | Request central processor. This puts the job in W status, and it will use the central processor if its priority is sufficient. The register settings of the exchange jump area will be used. |
| BKP,44300. | Breakpoint to address 44300 in the program. Central processor execution begins at the current value of P and stops when P = 44300. DIS clears 44300 to stop the program at that point, and restores the original word when the stop occurs. |
| RNS. | Read and execute next control statement. |
| RSS. | Read next control statement and stop prior to execution. |
| ENS.xxxxxxxxxxxxxx. | Allows the entry of any control statement xxxxxxxxxxxxxx as if it had been entered on a control card. The statement can then be processed using RNS. or RSS. |
| GO. | Restarts a program which has paused. |

ENTRIES FOR CHANGING JOB DISPLAY CONTENTS (cont'd)

| | |
|---|---|
| ONSW3. | Set sense switch 3 for the job. |
| OFFSW4. | Turn off sense switch 4 for the job. |
| HOLD. | DIS relinquishes the display console, but the job is held at the present status. A console must be reassigned to continue use of DIS. |
| DROP. | DIS is dropped and normal execution of the job is continued; it does not drop the job. |
| DMP(200,300) | Dump storage from 200 to 277 in the output file. |
| DMP(400) | Dump storage from the job reference address to 377. |
| DMP. | Dump exchange jump area to output file. (DMP formats are the same as if used on control cards). |

**DUMP STORAGE**

This peripheral program may be called from a display console with a control card in any of the forms shown below: An octal jump is entered in the output file with the central storage address and one data word per line.

DMP.

dumps the exchange area into the output file.

DMP, 3400.

dumps from the reference address to the parameter address.

DMP (4000, 6000)

dumps from the first address specified to the second.

MODE n.

This control card may be used to change the arithmetic exit mode. n is a single octal digit (See Exchange Jump Information, Section 3.3.) The exit mode is set to zero unless otherwise specified.

Example:

MODE 3.

EXIT.

The EXIT card can be used to separate the control cards associated with the normal execution of a job from a group of control cards to be executed in the event of an error exit as listed below:

| | |
|---|---|
| 1 TIME LIMIT. | Job has used all the central processor time it requested. |
| 2 ARITHMETIC ERROR. | Central processor error exit has occurred. |
| 3 PPU ABORT. | PP has discovered an illegal request, e.g., illegal file name or request to write outside job field length. |
| 4 CPU ABORT. | Central program has requested that the job be aborted. |
| 5 PP CALL ERROR | Monitor has discovered an error in the format of a PP call entered in RA+1 by a central program (can occur if a program accidentally writes in RA+1, as can condition 3). |
| 6 OPERATOR DROP. | Operator has requested the job be dropped. |
| 7 DISK TRACE LIMIT. | No more room on a disk unit used by a job. |

When one of these conditions occurs, an error flag (numbered as above) is set at the control point. In cases 1, 2, 5, 6, 7, a dayfile message is issued; and in case 3, the fault-finding PP issues a message (BUFFER ARGUMENT ERROR from CIO, or NOT IN PPLIB).

When an error flag is set, a search is made for the next EXIT control card; and if it is not found, the job is terminated. If an EXIT card is found, the error flag is cleared and succeeding control cards are processed. If an EXIT card is met and no error flag is set the job is terminated normally at that point.

Example:

| | |
|---|---|
| MYJOB,1,400,100000. | Job card |
| ASSIGN WT, TAPE1. | Request scratch tape |
| RUN. | Compile and execute |
| EXIT. | |
| DMP. | Dump exchange package |
| DMP,1000. | Dump first $1000_8$ words of store |
| 7,8,9 | End of control cards |
| (Program)<br>7,8,9 | |
| (Data)<br>6,7,8,9 | |

The dumps are made only if an error condition occurs.

Record Separator

This card, consisting of a 7,8,9 punch in column 1, separates the different types of records (control cards, source language cards, data cards) within a job.

File Separator

This card, consisting of a 6,7,8,9 punch in column 1, must be the last card of each job deck. No job may use information beyond this card.

| | |
|---|---|
| BKSP | Backspace medium |
| COPY | Copy to double file mark |
| COPYBF | Copy binary file |
| COPYBR | Copy binary record |
| COPYCF | Copy coded file |
| COPYCR | Copy coded record |
| COPYSBF | Copy shifted binary file |
| REWIND | Rewind medium |
| RETURN | Release equipment from control point assignment |
| RUN | Compile and run FORTRAN |
| UNLOAD | Rewind and unload medium |
| VERIFY | Verify two media |
| COPYX | Copy to Name |

Example: Two binary files on magnetic tapes are to be copied to disk as a single file called TAPE9 for use by a FORTRAN program which uses the punch.

```
THEJOB,10,1000,200000.      Job card
REQUEST FIRST.              Get first tape
REWIND(FIRST)
REQUEST SECOND              Get second tape
REWIND(SECOND)
COPYBF(FIRST, TAPE9)        First tape to disk
REWIND(FIRST)
BKSP(TAPE9)                 Backspace over file mark
COPYBF(SECOND, TAPE9)       Second tape to disk
REWIND(SECOND)
REWIND(TAPE9)
ASSIGN CP, PUNCH.
RUN.
7,8,9
    PROGRAM H3 (INPUT, OUTPUT, PUNCH, TAPE9)
    (Rest of Program)
7,8,9
    (Data)
6,7,8,9
```

Since TAPE9 has not been assigned specifically, it goes to disk.

FORTRAN Subroutines

| | |
|---|---|
| BACKSP | Backspace medium |
| CHAIN | Loads a program from the disk file and executes it. Parameters passed from one program to another must be in the common region. All segments to be chained must be compiled with the same file names. The segment name is transferred to the day file and displayed prior to execution. |
| DISPLA | Displays a variable name and its numerical value as an integer if unnormalized, in floating point format if normalized. |
| ENDFIL | Write end-of-file. |
| INPUTB | Binary input |
| INPUTC | FORTRAN data input |
| OUTPTB | Binary output |
| OUTPTC | FORTRAN data output |
| RANF | Random number generator |
| REWINM | Rewind medium |
| XLOCF | Returns the memory location of a variable name. |

# EQUIPMENT ASSIGNMENT

Any file not specifically assigned on control cards is assigned by the system to storage on disk unit zero. A job need not request card reader and printer for normal input/output since its cards are already stored in the job input file on disk, and output for a printer is sent to the job output file on disk. Input and Output files are normally stored on disk zero.

The control cards of a job are processed in order, so any equipment assignment must be made before the corresponding file is referenced.

**ASSIGN u, f**

This control card assigns any available peripheral unit of type u to a file named f. The type u may be any of the equipment listed below or it may be an equipment number, in which case, operator action is not required.

| | | | |
|---|---|---|---|
| DA | disk cabinet, channel 0 | CR | card reader |
| DB | disk cabinet, channel 1 | LP | line printer |
| DC | disk cabinet, channel 2 | MT | 607 magnetic tape (1/2") |
| DS | display console | WT | 626 magnetic tape (1") |
| CP | card punch | | |

This must be the first appearance of the name f in the job file. The file name f is alphanumeric, begins with a letter, and is a maximum of seven characters long. Multiple file names are not allowed.

Examples:

| | |
|---|---|
| ASSIGN50, TAPE6. | Assign equipment number 50 (channel 5, unit 0, 1/2" tape) to TAPE6. The job will be held up if this tape is presently assigned to another job. |
| ASSIGN CP, PUNCH. | Assign a card punch to the file named PUNCH. When the job writes to the file PUNCH, data will be punched on cards. |
| ASSIGN MT, TAPE2. | Operator to assign a 1/2" tape to file TAPE2. |
| ASSIGN WT, TAPE3. | Operator to assign a 1" tape to file TAPE3. |

For MT and WT a message for the operator is displayed under the number of the job's control point:

WAITING FOR MT (or WT)

To assign tape 61 to control point 6, the operator would key 6.ASSIGN61.

| | |
|---|---|
| ASSIGN01, TAPE9. | Use disk unit 1 to store file TAPE9. |

For TAPE, the ASSIGN statement is intended for scratch tapes only; the operator may assign any free tape of the specified type.

A user supplied tape should be assigned with a REQUEST statement.

# COMMON FILES

Common files are files that are not discarded upon job completion. Normally, input files used by a running job (type local) are dropped; disk space is freed or equipment released. Output files are printed and discarded after printing. A job may declare a file to be common so as to make it available to other jobs. However, a job to which a common file is attached, can change it to local if discarding is desired.

**COMMON f.**

This control card has two effects:

1. If the file name, f, has common status in the FNT/FST and is not being used by another job, it is assigned to this job until dropped. If the file is being used by another job or does not have common status, this job must wait until the file is available.

2. If the file name, f, already appears as a local file name for the job, the file will be assigned common status in the FNT/FST and is available to any succeeding job after it is dropped by this job.

A file generated by a job may not be declared in a COMMON card until the job has been completed.

Example: COMMON BFILE.

**RELEASE f.**

With this control card, the common file named f currently assigned to this job will be dropped from common status and assigned local status in the FNT/FST.

Example:

RELEASE BFILE.

The common file, named BFILE, attached to this job is changed to type local so that it will be dropped at the end of the job.

**REQUEST f.**

This control card requests the operator at the system display console to assign to this job the peripheral equipment specified by f. This must be the first appearance of the name f. The job waits for operator action before proceeding.

Example:

REQUEST TAPE 4.

Operator to assign an equipment for file TAPE 4.

In this case, the message REQUEST TAPE4. is displayed under the number of the job's control point, and the operator can key the number of the equipment on which the user's tape is mounted. For control point 4:
4. ASSIGN71.

## EXPORT/IMPORT

### I. MONITOR MESSAGES

1. (Name) In Job Stack
2. (Name) Assigned Control Point N
3. (Name) In Output Stack
4. (Name) Output Complete
5. Job Table Full
6. All Remote Jobs Complete
7. Communications With Central Lost

### II. OPERATOR DIRECTIVES

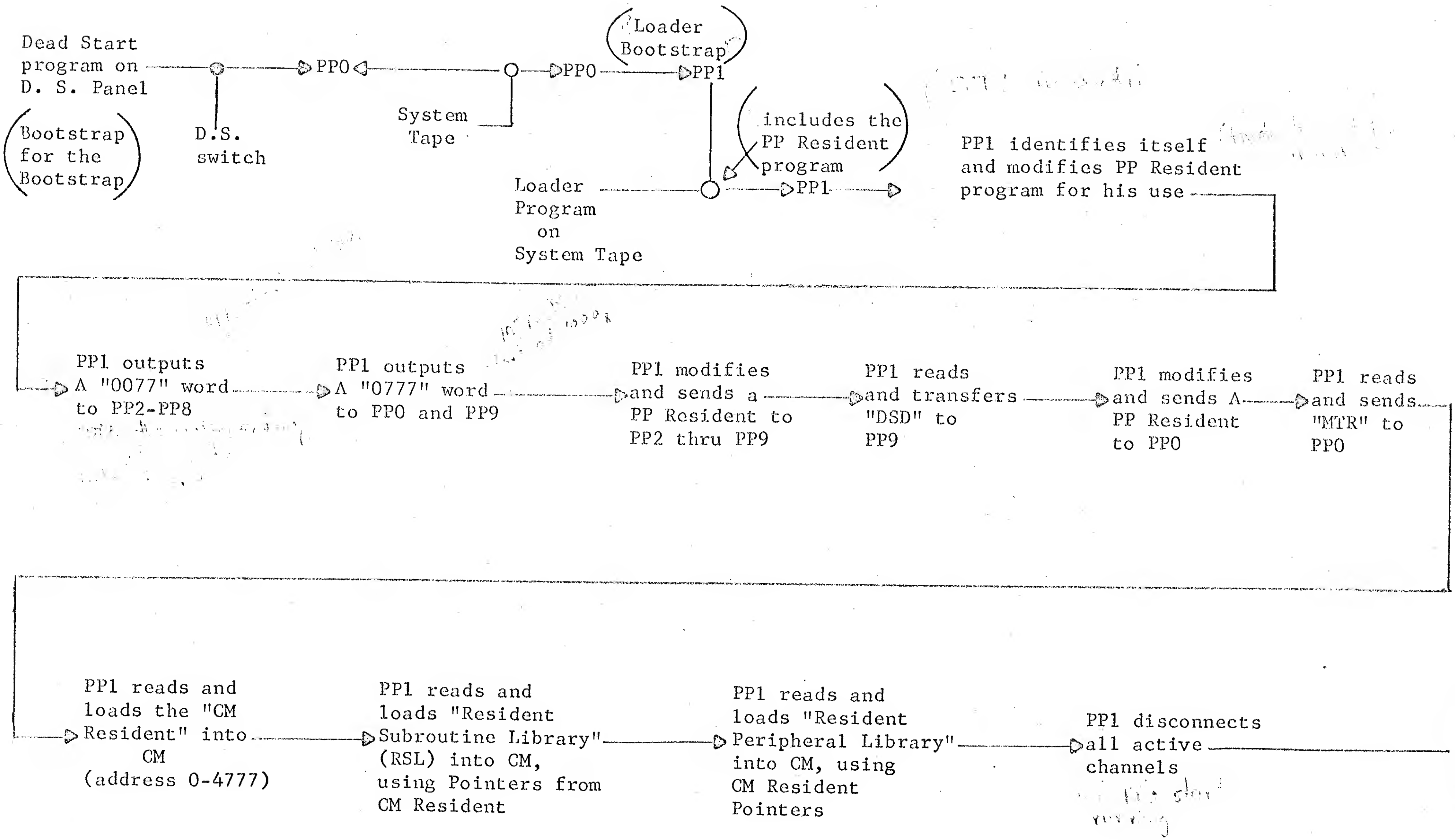
| | | |
|---|---|---|
| 1. | STAT, (Name) | Job Status Request |
| 2. | CPP, (Name), (Priority) | Change Priority |
| 3. | CPT, (Name), (Minutes) | Change Run Time |
| 4. | RPNT, N | Reprint N Sectors |
| 5. | RPCH, N | Repunch N Sectors |
| 6. | PRNT, N | Print N Copies |
| 7. | DVT, X, (Name) | Divert Output X = C   C→R<br>X = R   R→C |
| 8. | TERM, dv | Terminate Output |
| 9. | ABT, (Name), S | Abort Job |
| 10. | DISP, (Message) | Display Message |
| 11. | LIST | List Remote Jobs in System |
| 12. | END | Shut Down Communications |

### III. JOB STATUS RESPONSE

1. (Name) In Job Stack
2. (Name) Assigned Control Point (X,W,A,....)
3. (Name) In Output Stack
4. (Name) Not in System

# Dead Start Program Chippewa 1.1

| | Loc | | | | |
|---|---|---|---|---|---|
| | 0 | | ZEROS | | |
| | 1 | | 7513 | DCN | disconnect ch. 13 |
| | 2 | Mode I | 7713 | FNC | select equip #5 on ch 13 |
| | 3 | | 5000 unit # | | unit 0 [usually that's where MT is] |
| | 4 | Mode I | 7713 | FNC | Rewind selected equipment |
| rewind is 206X - p-31 codes | 5 | | 0010 look this up in 3000 code manual | | |
| | 6 | Mode II | 7713 | FNC | Read binary to-end-of-record |
| | 7 | | 1400 | | |
| | 10 | | 7413 | ACN | Activate ch. 13 |
| | 11 | | 7113 | IAM | input from ch. to block |
| | 12 | | 0015 | | beginning at PP loc. 15 |
| | 13 | | Play | various purposes e.g. MACE, SMM | |
| | 14 | | Play | | |
| | 15 | | ZERO | | |




C.O.S DEAD START, SIMPLIFIED, PART I 7/23/67

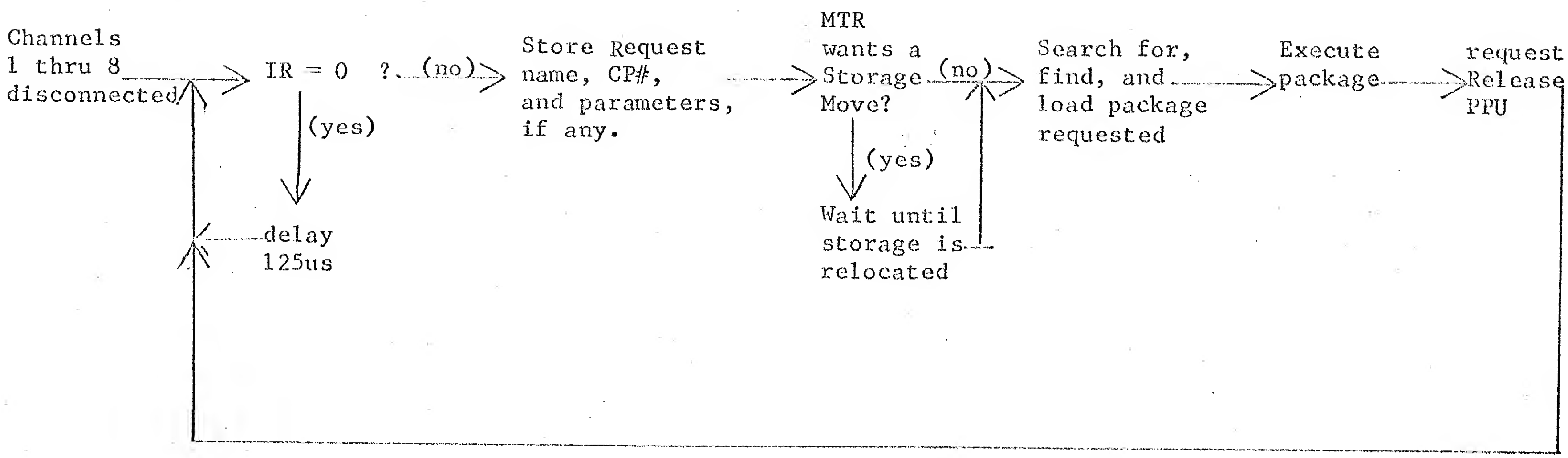
PP0 (MTR) assigns next available track to the 1st FNT/FST entry (DAYFILE) → MTR Exchange Jumps CPU into Idle Program (at Control Point 0 Stop instruction, P=0-02.) → MTR enters its Master Loop and processes Requests

PP9 (DSD) initiates itself, selects "A" screen on left tube, "B" screen on right tube → DSD requests MTR to Reserve Channel 10 → DSD enters its Master Loop, paints "A" & "B" screens, looks for Keyboard entries and sets up delay for next loop.

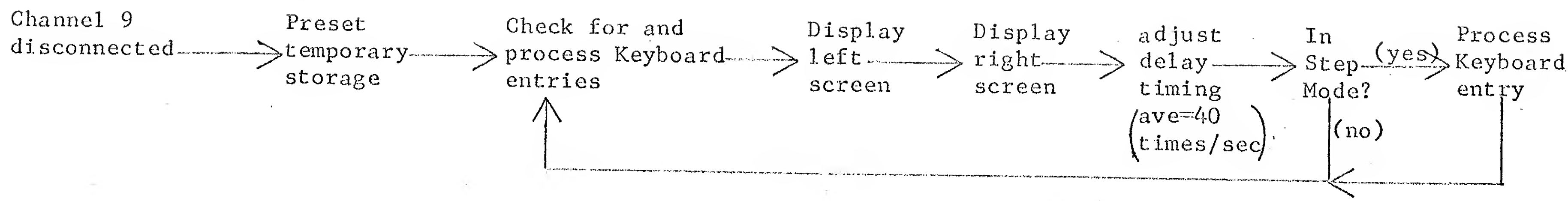
PP1 requests an available track from MTR → PP1 reads and transfers the rest of the System Tape to disk, recording the name and disk location of each package in the CM Resident Directories. → PP1 enters its PP Resident Idle Loop

PP2 thru PP8 enter their PP Resident program "Idle Loop"

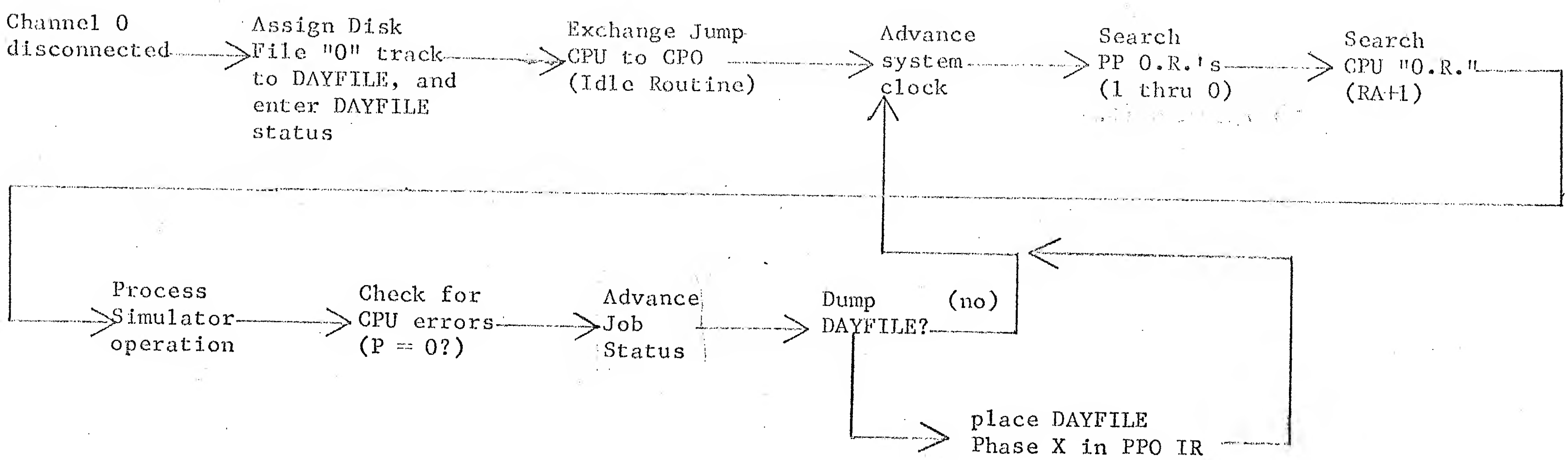
System waits for operator directions via the keyboard entry




PPR (PP RESIDENT) MASTER LOOP, SIMPLIFIED 4/17/67




DSD MASTER LOOP, SIMPLIFIED 1/13/67




MTR MASTER LOOP, SIMPLIFIED 1/13/67

Operator enters "AUTO. (cr)"

→

DSD recognizes a non-zero Keyboard input

→

Upon recognizing a (cr), DSD translates the entry, "AUTO".

→

DSD places "ILJ" and CP"1" in its Message Buffer and requests MTR to "ASSIGN PPU" (Function 20 in its O. R.)

→

MTR finds the Function 20 in PP9's (DSD) O.R., finds a free PPU, assigns it to CP1, and copies word one of DSD's Message Buffer to that PP's I.R.

→

MTR clears out PP9's O.R. and continues in its Master Loop

→

DSD finds its O.R. empty and places "1OT" and CP"2" in its Message Buffer, and requests MTR with a Function "20"

→

MTR processes this request like the 1st

→

DSD writes "1BJ" & CP"3", requests a PP.

→

MTR processes this request like 1st

→

DSD writes "1BJ" & CP"4" like 1st

→

MTR processes this request like 1st

→

DSD writes "1BJ" & CP"5" like 1st

→

MTR processes this request like 1st

→

DSD writes "1BJ" & CP"6" like 1st

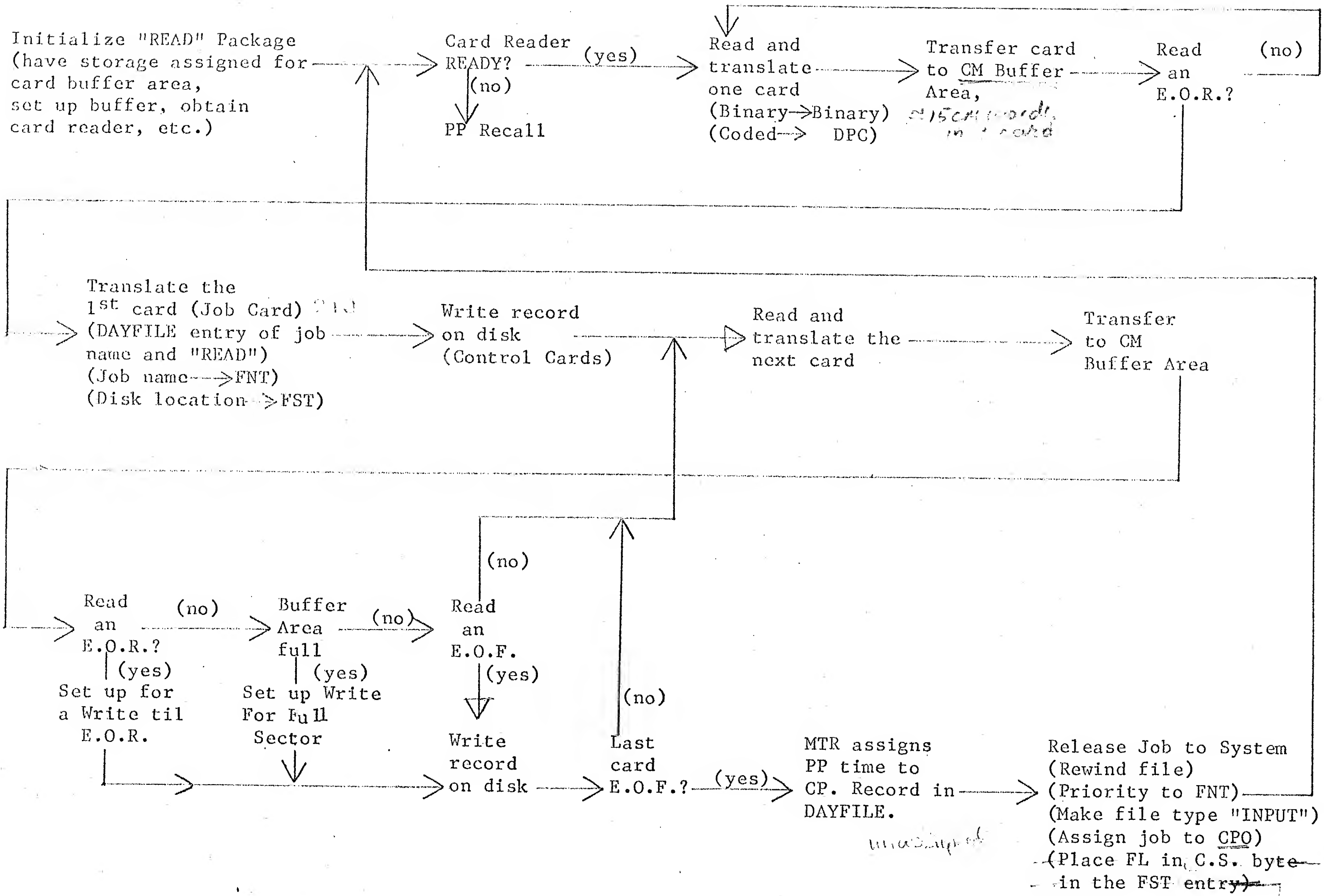
→

MTR processes this request like 1st

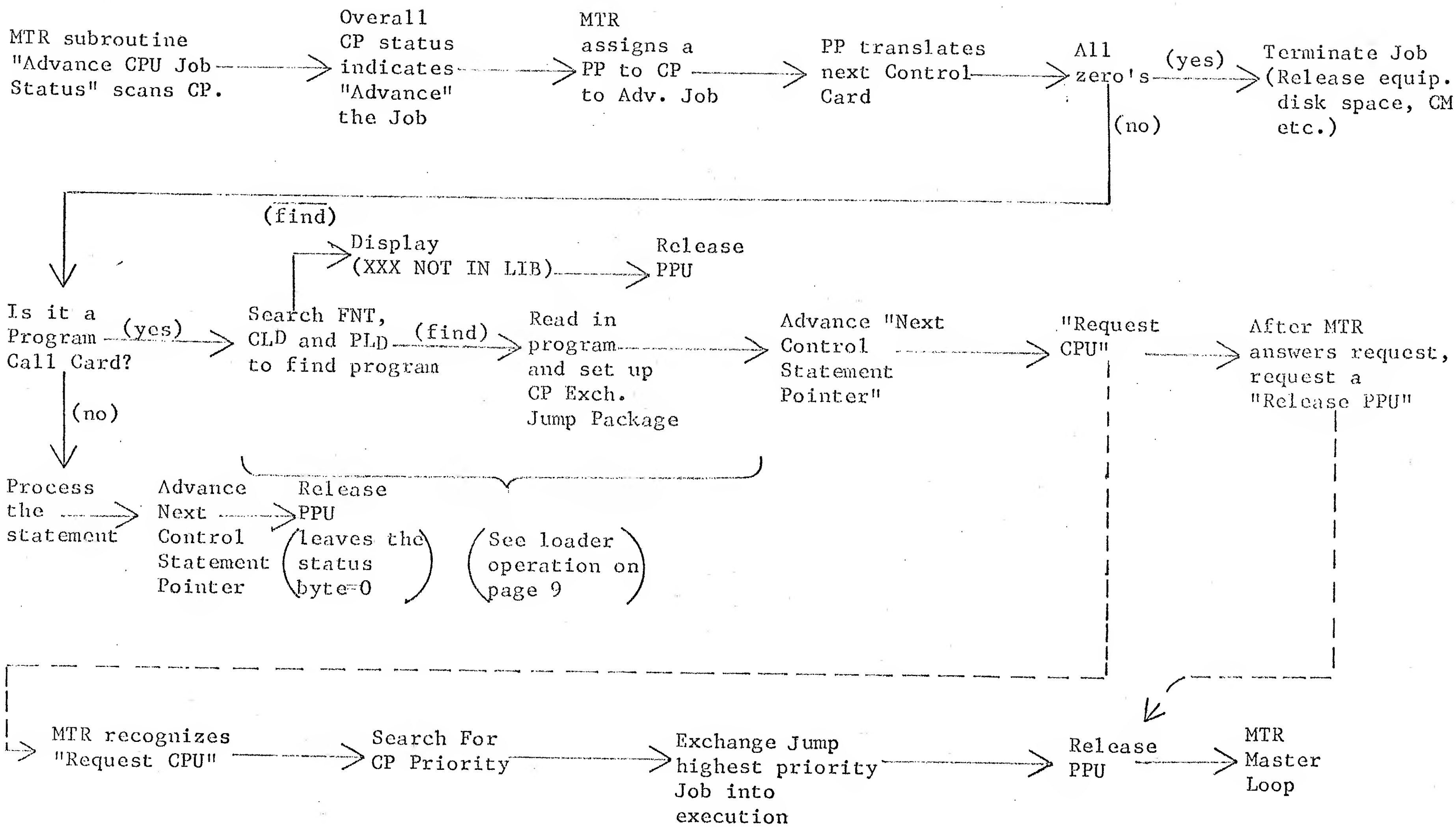
→

DSD returns to Master Loop

C.O.S., "AUTO. (CR)", SIMPLIFIED, PART I 4/17/67




C.O.S., JOB LOAD, SIMPLIFIED 4/19/67




C.O.S. ADVANCE JOB, SIMPLIFIED 4/17/67

Initialize "NEXT" package → Searches for the highest priority, unassigned, type "INPUT" file with an FL that will fit in CM.

(no find) → Job name = NEXT
Console msg = IDLE
Enter PP Recall → Release PPU

(find) → Enter job name in CP Area. Obtain disk location from FST → Read 1st record from disk, then transfer into "Control Statement Buffer" at CP Area

→ Translate 1st Control Card (Job Card) → Job Card parameters to CP Area (Allocate CM) etc. → Advance "Next Control Statement Pointer" → Request Release PP → MTR releases PP → *

* This will leave the CP status byte = 0——— 0, and the Recall register equal to 0———0. Also the CPU has not yet been called, thus the CP address is not in the CPU Job Stack. These conditions will indicate to MTR to "Advance" the Job.

C.O.S., BEGIN JOB, SIMPLIFIED 4/17/67

1AJ calls 2TS → 2TS translates Program Call Card → 2TS enters parameters for load operation into RA through RA + $77_8$ → 2TS calls "LOD" (through PPR) → "LOD" overlays 1AJ & 2TS (0773→ 2256) → LOD searches RSL & CLD for "LOADER" → LOD loads "LOADER" into RA + FL - the loader length and up.

→ Set P in exchange jump package to enter LOADER program at "LOADER 1" → LOD calls "LDR" through PPR (0773-7632) → Exit into "LDR" → LDR calls 2LA to check the parameters → 2LA calls 2LB to restore LDR → LDR loads the named file into CM, RA + 100 and up.

→ Request CPU (to start LOADER) → Request Release PPU — — — CPU is exch. jumped into LOADER → LOADER finds out main program is loaded → CPU jumps to main program — — → Program needs next Segment or Overlay

→ CPU jumps to LOADER with parameters for next Segment or Overlay → LOADER calls "LDR" to load next Segment → Is LDR finished loading? (no) → Recall → (back to LOADER calls "LDR")

Is LDR finished loading? (yes) → LOADER links Segment or Overlay to main program → CPU jumps to main program — ( etc. ) → CPU calls "END" → Stop CPU

6000 SCOPE 2.0 "SIMPLIFIED" LOADER OPERATION 4/17/67

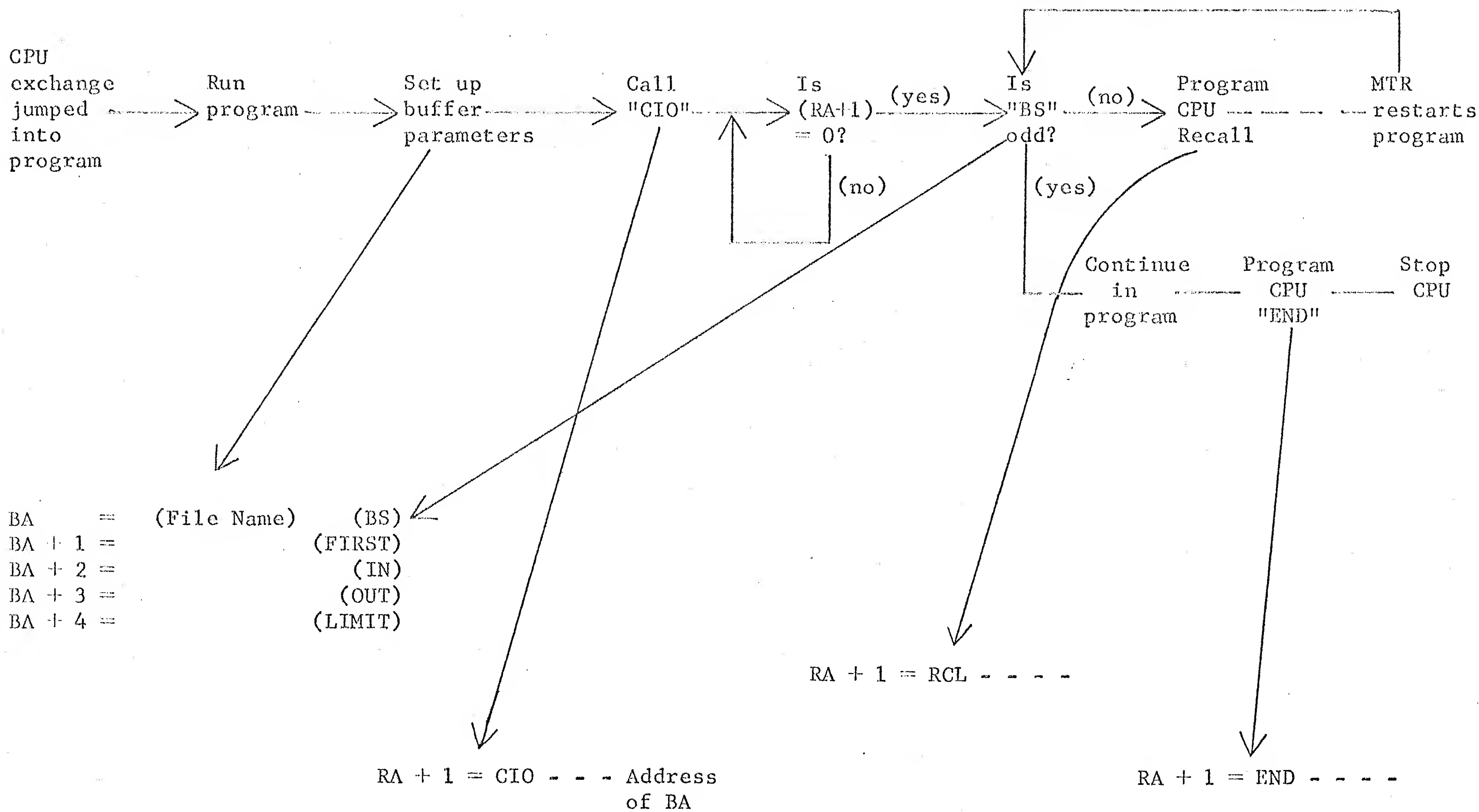
SEGMENTS

| LOADING ORDER | SEGMENT LEVEL | CONTENTS OF USER'S JOB AREA IN MEMORY AFTER LOADING OF SEGMENT | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | SEG 0 | UNUSED STORAGE AREA | | | |
| 2 | 3 | SEG 0 | SEG 3 | | | |
| 3 | 4 | SEG 0 | SEG 3 | SEG 4 | | |
| 4 | 9 | SEG 0 | SEG 3 | SEG 4 | SEG 9 | |
| 5 | 2 | SEG 0 | SEG 2 | | | |
| 6 | 1 | SEG 0 | SEG 1 | | | |
| 7 | 5 | SEG 0 | SEG 1 | SEG 5 | | |
| 8 | 8 | SEG 0 | SEG 1 | SEG 5 | SEG 8 | |
| 9 | 7 | SEG 0 | SEG 1 | SEG 5 | SEG 7 | |

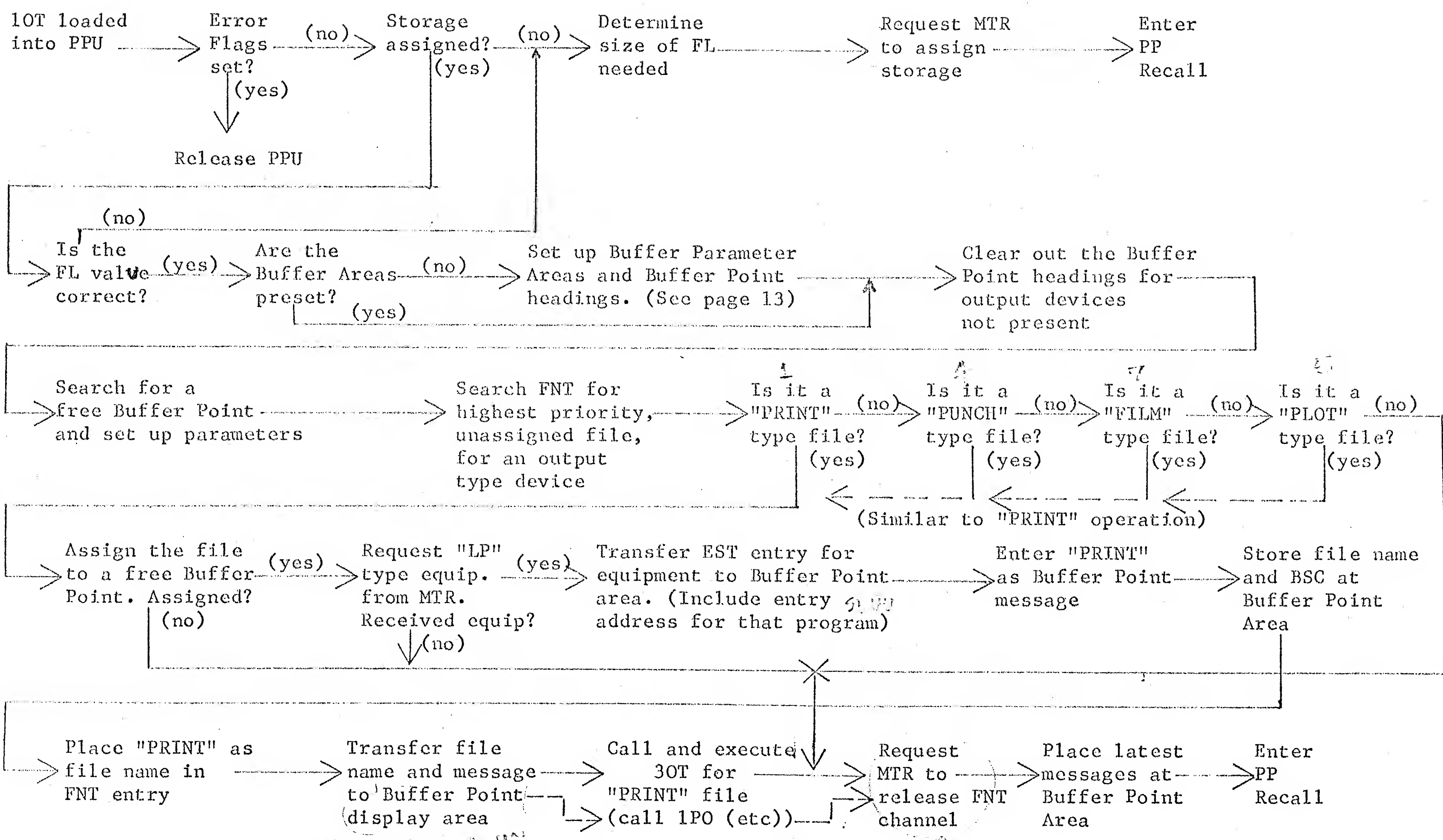
OVERLAYS

| LOADING ORDER | PRIMARY LEVEL NUMBER | SECONDARY LEVEL NUMBER | CONTENTS OF USER'S JOB AREA IN MEMORY AFTER LOADING OF OVERLAY | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | (0,0) | UNUSED STORAGE AREA | |
| 2 | 1 | 0 | (0,0) | (1,0) | |
| 3 | 1 | 1 | (0,0) | (1,0) | (1,1) |
| 4 | 1 | 2 | (0,0) | (1,0) | (1,2) |
| 5 | 1 | 1 | (0,0) | (1,0) | (1,1) |
| 6 | 1 | 3 | (0,0) | (1,0) | (1,3) |
| 7 | 1 | 2 | (0,0) | (1,0) | (1,2) |
| 8 | 2 | 0 | (0,0) | (2,0) | |
| 9 | 2 | 1 | (0,0) | (2,0) | (2,1) |
| 10 | 2 | 2 | (0,0) | (2,0) | (2,2) |
| 11 | 3 | 0 | (0,0) | (3,0) | |
| 12 | 4 | 0 | (0,0) | (4,0) | |

overlays are not relocatable




C.O.S. CPU PROGRAMMED INPUT/OUTPUT 4/17/67




C.O.S./SCOPE 2.0 "OUTPUT" SIMPLIFIED FLOW CHART 8/28/67

| Address | Contents | | |
|---|---|---|---|
| RA + 0 - 7 | Not used | | |
| RA + 10 | File Name | Buffer Status Code | Buffer Point #1's Parameter Area |
| RA + 11 | N/U | FIRST | |
| RA + 12 | N/U | IN | |
| RA + 13 | N/U | OUT | |
| RA + 14 | N/U | LIMIT | |
| RA + 15 | Job Name | N/U | |
| RA + 16 | Program Entry Addr. \| B A \| D C \| Bfr.Pt.#="X" 0 0 X 0 | S E U U | |
| RA + 17 | Buffer Point Time Accumulation → | | |
| RA + 20 - 27 | Buffer Print #2's Parameter Area | | |
| RA + 30 - 37 | Buffer Point #3's Parameter Area | | |
| RA + 40 - 47 | Buffer Point #4's Parameter Area | | |
| RA + 50 - 57 | Buffer Point #5's Parameter Area | | |
| RA + 60 - 67 | Buffer Point #6's Parameter Area | | |
| RA + 70 - 77 | Not used | | |
| RA + 100 through RA + 1077 | Buffer Point #1's Buffer Area | | |
| RA + 1100 through RA + 2077 | Buffer Point #2's Buffer Area | | |
| RA + 2100 through RA + 3077 | Buffer Point #3's Buffer Area | | |
| RA + 3100 through RA + 4077 | Buffer Point #4's Buffer Area | | |
| RA + 4100 through RA + 5077 | Buffer Point #5's Buffer Area | | |
| RA + 5100 through RA + 6077 | Buffer Point #6's Buffer Area | | |

C.O.S./SCOPE 2.0 "OUTPUT" PARAMETER/BUFFER AREA 8/28/67

CONTROL DATA
CORPORATION

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

PRINTED IN U.S.A.